

Bernd-Holger Schlingloff

Formale Methoden in der Praxis

Softwaredesign für Luft- und Raumfahrt

Antrittsvorlesung

16. Mai 2002

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Die digitalen Ausgaben der *Öffentlichen Vorlesungen* sind abrufbar über den Dokumenten- und Publikationsserver der Humboldt-Universität unter: <http://edoc.hu-berlin.de>

Herausgeber:

Der Präsident der Humboldt-Universität zu Berlin

Copyright: Alle Rechte liegen beim Verfasser

Berlin 2005

Redaktion:

Birgit Eggert

Forschungsabteilung der Humboldt-Universität zu Berlin

Unter den Linden 6

D–10099 Berlin

Herstellung:

Forschungsabteilung der Humboldt-Universität zu Berlin

Unter den Linden 6

D–10099 Berlin

Heft 138

ISSN 1618-4858 (Printausgabe)

ISSN 1618-4866 (Onlineausgabe)

ISBN 3-86004-187-8

Gedruckt auf 100 % chlorfrei gebleichtem Papier

Sehr geehrte Damen und Herren, lieber Herr Prof. Björner,

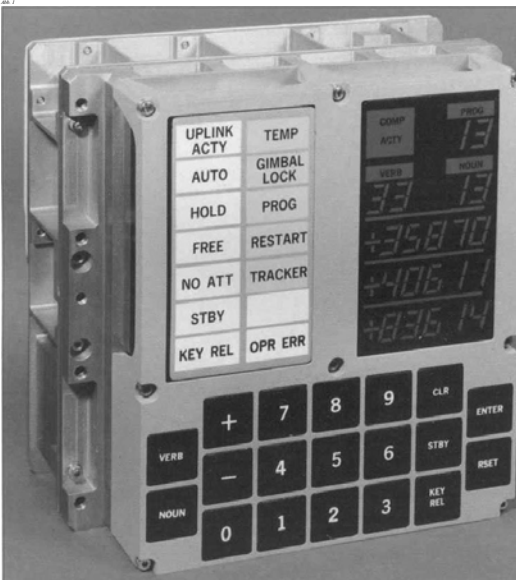
in der heutigen Vorlesung möchte ich einen speziellen Bereich der Softwaretechnik darstellen, nämlich die Anwendung formaler Methoden zur Entwicklung sicherheitskritischer Software. Wir bezeichnen dabei ein Programm als *sicherheitskritisch*, wenn es bei Fehlfunktionen Personenschaden oder großen materiellen Schaden verursachen kann. Typische Beispiele sind Softwarelösungen in der Medizintechnik, der Servicerobotik oder im Transportwesen; aber auch im Mobilfunk oder ganz allgemein in Bereichen mit hohen produzierten Stückzahlen kann ein Softwarefehler erhebliche Kosten verursachen. Das Anwendungsgebiet, welches nachfolgend im Vordergrund steht, sind Systeme der Luft- und Raumfahrt.

1 Probleme in Flugsoftware: drei historische Beispiele

Ein großer Teil der Software in diesem Bereich wird für Steuercomputer („embedded controller“) entwickelt. Üblicherweise sind diese Systeme sicherheitskritisch (nach unserer obigen Definition); oft hängt das Gelingen einer Mission von der korrekten Funktionsweise aller beteiligten Komponenten ab. Darüber hinaus bieten solche Systeme viele interessante Probleme und Fragestellungen, die bislang softwaretechnisch nicht hinreichend beherrscht werden: Flugsoftware ist oft reaktiv (d.h. interagiert kontinuierlich mit der Umgebung), realzeitbezogen (d.h. ist durch eine äußere Uhr bestimmt), hardwareabhängig (d.h. funktioniert nur für eine bestimmte Plattform und Konfiguration) und für eine bestimmte Aufgabe maßgeschneidert (d.h. wird nicht nach gängigen Konventionen entwickelt). Als Vorteil erweist sich dabei, dass die während einer Mission aufgetretenen Probleme in der Regel gut dokumentiert sind, und daher gut untersucht werden können.

In gewisser Hinsicht kann die Raumfahrt sogar als die treibende Kraft bei der Entwicklung heutiger Computer bezeichnet werden. Während die Wurzeln des PCs sicherlich in den Dechiffrier-

maschinen und ballistischen Kalkulatoren des zweiten Weltkrieges liegen, waren diese Rechenmonster für einen Einsatz an Bord eines Flugobjektes viel zu schwer. Historische Tatsache ist, dass die Miniaturisierung von Computern in den 1960-ern durch Raumfahrtaufträge vorangetrieben wurde. Zum Beispiel verwendete der Bordcomputer im Apollo-Projekt erstmalig die neu entwickelten „integrierten Schaltkreise“. Es gehört zur Folklore, dass eben dieser Bordcomputer der Apollo 11-Mission beim Landeanflug der ersten Mondlandung (am 24. 7. 1969) versagte, woraufhin der Kommandant der Landefähre, Edwin Aldrin, ihn „abschaltete“ und die Eagle „von Hand“ landete.

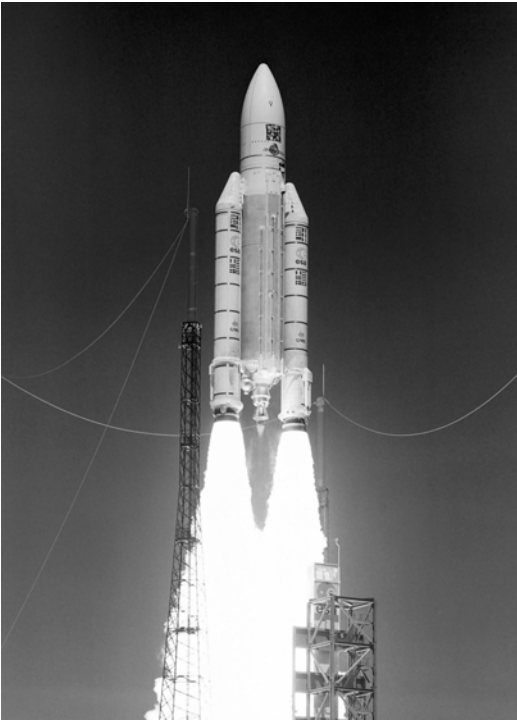


*Abb. 1:
Apollo-11 Bordcomputer
(© Raytheon)*

In der Abb. 1 sehen Sie diesen Bordcomputer: Beachten Sie vor allem das ergonomische Eingabefeld, mit welchem Prozesse durch die Eingabe ihrer Nummer (!) von den Astronauten gestartet werden konnten. Was war aber nun während der Landung wirklich passiert? In [1] findet sich eine ausführliche Darstellung der Vorgänge. Die völlig neu entwickelte Technologie der so genannten „integrierten Schaltkreise“ war vielfach störungsanfällig, insbesondere bei:

- Spannungsschwankungen,
- Taktstörungen,
- Interrupt-Blockaden, oder
- Betriebssystemfehlern.

Als Lösung hatte man einen schnellen Neustartmechanismus realisiert, mit dem der Computer seine Arbeit nach einer Störung an der Stelle wieder aufnehmen sollte, an der er unterbrochen worden war. Dazu führte jeder Prozess eine Phasentabelle mit, in der sein jeweiliger Wiedereinstiegspunkt verzeichnet war. Ein einmal gestarteter Prozess wurde kontinuierlich so lange ausgeführt, wie er rechenbereit war; heute bezeichnet man dies als kooperatives Multitasking. Für das Betriebssystem war ein unabhängiger Wachhund implementiert worden, der es in regelmäßigen Abständen unterbrach. Damit einfache Zählerprozesse, die nur den Wert eines Register erhöhen oder erniedrigen sollten, nicht übermäßig lange warten mussten, waren diese höher priorisiert als die lang laufenden Benutzerprozesse und in einer besonderen Warteschlange. Man hatte im Voraus geschätzt, dass maximal 85.000 Inkrement- oder Dekrementprozesse pro Sekunde möglich sein sollten, ohne dass der Rechner überlastet wird. Im Landeanflug nun erzeugte der Radarsensor für das Rendezvous so viele Interrupts, dass 15% der Prozessorleistung beansprucht wurden. Die Ursache dafür war, dass ein Analog-Digital-Wandler wegen einer falschen Einstellung „flimmerte“ (ständig zwischen zwei benachbarten Werten hin- und hersprang und jedes Mal einen Zählerprozess in Gang setzte). Die Unterbrechungsbehandlung war für diese vielen Prozesse zu langsam: einige Kontrollroutinen wurden neu gestartet, bevor sie überhaupt zu Ende abgearbeitet werden konnten. Dadurch wurde der Neustart ausgelöst, und zwar dreimal binnen 40 Sekunden. In der Bodenkontrollstation wurde das Problem an Hand der Meldungen analysiert und erkannt. Es wurde beschlossen, die Landung fortzusetzen und die Landungsroutine P66 zu starten (durch Eingabe ihrer Nummer, wie oben geschildert). Dadurch wurde das Problem buchstäblich in letzter Minute gelöst und der Stoff für ungezählte Computerspiele der ersten Generation erschaffen. Immerhin wurden einige Konsequenzen aus diesem Versagen



*Abb. 2a:
Jungfernflug der
Ariane 5
(© ESA /CNES/
ARIANESPACE-
SERVICE Optique
CSG)*

gezogen, die auch heute noch eingehalten werden müssen: Der Prozessor darf in kritischen Systemen nur bis zu einer vorgegebenen Maximallast (deutlich unter 100 %) ausgelastet werden, Sensoren werden grundsätzlich redundant ausgelegt, und Prioritäten dürfen nur sehr restriktiv vergeben werden.

Ein anderes, sehr bekanntes Beispiel für einen Softwarefehler in der Raumfahrt war die Explosion der Ariane 5 auf ihrem Jungfernflug 501 am 4. 6. 1996 (Abb. 2a, 2b, S. 7). Bei diesem Flug wurde 37 Sekunden nach dem Start eine Kursabweichung gemessen, nach 40 Sekunden explodierte die Rakete. Auch dieser Fehler wurde sehr gut untersucht und dokumentiert: nach dem Absturz wurden mit großem Aufwand die Flugschreiber aus den Sümpfen bei Kourou geborgen und analysiert. Der ausführliche Abschlussbericht der eingesetzten Untersuchungskommission wurde im Internet publiziert [2]. In diesem Bericht wird die Fehlerursache



Abb. 2b:

Explosion der Ariane 5 auf ihrem Jungfernflug 501 am 4.6.1996

(© N. Lecore/GAMMA for ESA/CNES)

eindeutig festgestellt; die Kausalkette für den Absturz wird in umgekehrter Reihenfolge aufgelistet:

- 40 Sekunden nach dem Start explodierte die Rakete, weil der automatische Selbstzerstörungsmechanismus ausgelöst worden war (Sprengung der Rakete).
- Die Sprengung war notwendig geworden, weil die Rakete auf Grund ihrer Schräglage drohte, auseinander zu brechen.
- Die Schräglage der Rakete kam daher, dass die Ruder falsch eingestellt waren, weil sie vom Bordcomputer fehlerhaft angesteuert worden waren.
- Der Bordcomputer hatte versucht, mit der extremen Ruderposition eine vermeintliche Fehlbahn der Rakete auszugleichen, die er auf Grund falscher Lagedaten errechnet hatte.
- Die fehlerhaften Lagedaten waren von einer Lagebestimmungseinheit übermittelt worden, nachdem diese ausgefallen waren.
- Die Lagebestimmungseinheiten waren ausgefallen, weil bei der Übermittlung eines Messwertes an die Bodenstation ein

Übertragungsfehler aufgetreten war, der auf einen Zahlbereichsüberlauf zurückzuführen war (Real nach Int-Konversion).

- Der Wert der entsprechenden Messgröße war vor der Konversion nicht überprüft worden, da die Entwickler stillschweigend angenommen hatten, dass die Beschleunigung immer innerhalb eines bestimmten Intervalls liegen würde. Dieser Fehler war durch die nachfolgende Qualitätssicherung nicht entdeckt worden, da die Annahme nicht explizit aufgeschrieben worden war.
- Die Annahme war nicht dokumentiert worden, da sie bei der Ariane 4, für die die Lagebestimmungseinheit ursprünglich konstruiert worden war, trivialerweise immer erfüllt war. Aus „Sicherheitsgründen“ war das in der Ariane 4 schon hundertfach bewährte Bauteil unmodifiziert für die Ariane 5 übernommen worden.

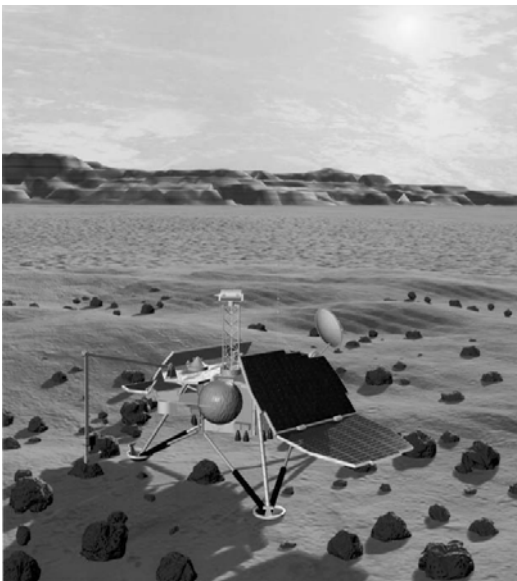
Eine Ironie des Schicksals war es, dass die fragliche Lageregelungseinheit sogar doppelt („fehlertolerant“) vorhanden war. Da der Fehler allerdings nicht zufällig, sondern systematisch bedingt war, fielen beide Repliken zur exakt selben Zeit aus. Die fatale Konversion des reellen Messwertes in eine ganzzahlige Größe für die Übertragung zur Bodenstation war außerdem nur zu Diagnose- und Telemetrie-zwecken notwendig gewesen; es hätte ausgereicht, den Wert intern zu speichern und nur auf Anfrage abzurufen. Ein weitere Ironie war, dass der Zahlbereichsüberlauf in einer Kalibrierungsroutine auftrat, die eigentlich nur vor dem Start von Bedeutung war (Pre-Launch). Der eigentliche Sinn der Routine war es, bei eventuellem Startabbruch die Wartezeit bis zum erneuten Countdown zu verkürzen. Wiederum aus „Sicherheitsgründen“ hatte man beschlossen, die Routine auch nach dem Start weiterlaufen zu lassen, um während der kritischen Phase so wenig wie möglich an der Softwarekonfiguration zu ändern.

In ihrem Bericht enthüllte die Kommission eine ganze Reihe von Gründen, die zu dem Fehler führten. Ein direkter Grund war sicherlich die explizite Entscheidung, nur vier der sieben reellen

Variablen in der betreffenden Routine vor der Konversion gegen einen eventuellen Bereichsüberlauf zu schützen. Ein weiterer war die explizite Entscheidung, den Lageregelungsalgorithmus im Fehlerfall abzurechnen und nicht „so gut wie möglich“ weiterlaufen zu lassen, genauso wie die Entscheidung, einen nicht benötigten Kalibrierungsprozess nach dem Start weiterlaufen zu lassen. Aber auch der gesamte Softwareentwicklungsprozess, bei dem undokumentierte Anforderungen die Qualitätssicherung und Systemintegration passieren, war Gegenstand der Kritik.

Der Bericht der Kommission führte zu massiven Änderungen in den beteiligten Softwareentwicklungsabteilungen. Insbesondere wurde das Qualitätsbewusstsein deutlich gehoben und ein größeres Gewicht auf formale Korrektheitsbeweise gelegt. In folgenden Starts gab es keine (bekannt gewordenen) Softwareprobleme.

Das dritte Beispiel, dass ich in der heutigen Vorlesung anführen möchte, ist der „Mars Polar Lander“ (MPL) der NASA, der am 3. 12. 1999 auf der Marsoberfläche verschollen ist. Die Aufgabe



*Abb. 3:
Mars Polar Lander
(© NASA/JPL)*

dieser Sonde war es, (nahe) am Südpol des Mars zu landen und 90 Tage lang geologische und klimatographische Messdaten von der Umgebung der Landestelle zu übertragen. In Abb. 3 (S. 9) ist eine Fiktion der erfolgreich gelandeten Sonde auf dem Mars dargestellt.

Die Mission erfolgte im Zeichen der „faster, cheaper, better“-Politik der NASA, mit der versucht werden sollte, die extrem hohen Kosten der Entwicklung von Raumfahrtsoftware zu senken. Das Gesamtbudget betrug nicht einmal die Hälfte von dem des „Mars Pathfinder“, bei höheren technologischen und zeitlichen Anforderungen. Daher wurde zum Beispiel aus Kostengründen darauf verzichtet, während der Landung Telemetriedaten zu übertragen, da dies zusätzliche Steuermanöver erfordert hätte. Das Ergebnis war, dass sich der MPL nach der Abtrennung von der Transportstufe und dem Eintritt in die Marsatmosphäre nicht mehr meldete und spurlos verschollen blieb. Alle Versuche, eine Verbindung zur Sonde herzustellen, schlugen fehl, und die Mission wurde am 17. 1. 2000 für gescheitert erklärt.

Das eingesetzte Untersuchungskomitee vom Jet Propulsion Laboratory [JPL 00] führte eine so genannte „Failure modes and effect analysis“ (FMEA) durch, bei der mögliche Ausfallgründe und ihre Auswirkungen während jeder Phase des Fluges analysiert wurden. In Abb. 4 ist eine Übersicht (aus [3]) über alle diese Fehlerursachen angegeben.

Die Untersuchungen ergaben, dass die wahrscheinlichste Ursache für den Ausfall der Sonde ihr Absturz aus 40 Metern Höhe war. Der MPL hatte im Landegestell Magnetsensoren integriert, die feststellen sollten, ob die Sonde auf dem Boden aufgesetzt hatte. Beim Bodenkontakt sollten die Bremstriebwerke von der Software abgeschaltet werden. Mehrere Experimente mit diesen Sensoren bewiesen, dass mit großer Wahrscheinlichkeit bereits beim Ausfahren der Füße (in 1500 Metern Höhe, während die Sonde noch am Fallschirm hing) ein solches Aufsetzsignal erzeugt wurde. Die Software beachtete dieses Signal als gültiges Anzeichen der Landung und schaltete die Landetriebwerke 40 m

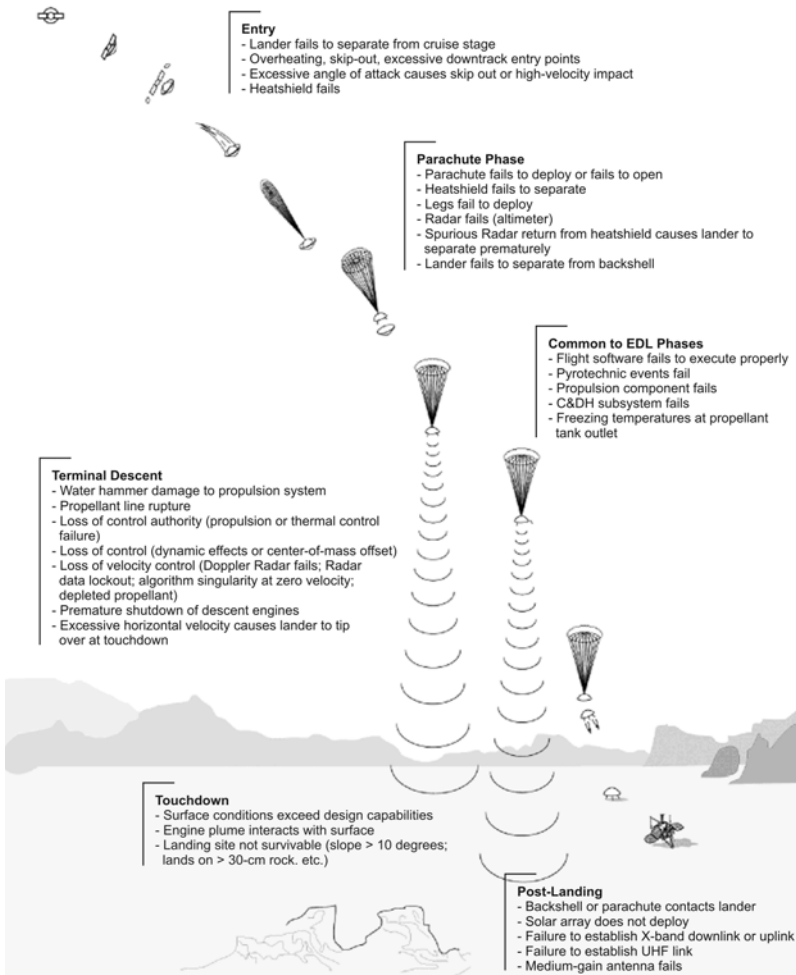


Abb. 4:
 Übersicht der Fehlerursachen
 (© NASA/JPL)

über dem Boden ab. Trotz der Bremsfallschirme ergab das eine Aufprallgeschwindigkeit von ca. 22 m/s oder ca. 80 km/h, die die Sonde sicherlich total zerstörte.

Die überschüssigen Sensorsignale waren im Entwurf vorausgesehen worden: Die Systemspezifikation besagte, dass „Touchdown“-Signale erst nach Erreichen von 40 Metern Höhe beachtet werden sollten. Die Routine zur Überwachung des „Touchdown“ wurde von der Flugüberwachungssoftware nach der Eintrittsphase gestartet und dann in 40 Metern Höhe aktiviert, um die Prozessorlast konstant zu halten. Durch das Ausfahren der Füße wurden jedoch bereits vor der Aktivierung überschüssige Signale generiert und in der entsprechenden Variablen „IndicatorState“ vermerkt. In der Software wurde versäumt, diese Variable vor der Aktivierung zurückzusetzen, daher führte die Aktivierung direkt zum Ausschalten der Triebwerke. Obwohl die Möglichkeit überschüssiger Sensorsignale von den Designern einkalkuliert worden war, waren diese in der Spezifikation nicht erwähnt worden. Daher wurde die eben genannte Anforderung bei der Umsetzung der Systemspezifikation in eine Anforderungsspezifikation nicht weiter berücksichtigt. Auch hier war also die letztendliche Fehlerursache eine mangelhafte Spezifikation.

Der Fehler hätte vermieden werden können, wenn die Ergebnisse der Integrationstests für das Landegestell rechtzeitig an die Verifikationsgruppe weitergegeben worden wären. Aufgrund der extrem hohen Zeit- und Budgetrestriktionen waren die allgemeinen Qualitätssicherungsmaßnahmen jedoch unzureichend. Die Mitglieder des Sensorikteams hatten nicht die Möglichkeit, den Code zu kontrollieren; die meisten Entwickler waren mindestens 60, einige bis zu 80 Stunden pro Woche mit Programmieren beschäftigt. Der Bericht resümiert: „In short, there was insufficient time and workforce available to provide the levels of checks and balances normally found in [space] projects.“

Generelle Beobachtungen aus diesen drei Beispielen sind, dass in Luft- und Raumfahrtapplikationen meist ein komplexes Zusammenspiel von Hard- und Software vorzufinden ist. Fehler ergeben sich meist in der Kombination von unvorhergesehenen Ereignissen, oft sind Randbedingungen nicht genügend spezifiziert. Obwohl eine Wiederverwendung von Komponenten angestrebt wird, ist diese problematisch, solange die Schnittstellen

nicht vollständig spezifiziert sind. Für solche Systeme mangelt es an realistischen Erprobungsmöglichkeiten: Der erste Einsatz ist oftmals schon der Ernstfall. Und schließlich hat die Branche inzwischen wie alle anderen mit einem erheblichen Termin- und Kostendruck zu leben, obwohl die Qualitätsanforderungen ungleich höher sind als in anderen Bereichen.

Der bei der NASA derzeit praktizierte Lösungsansatz für diese Herausforderungen heißt „IV&V“, unabhängige Verifikation und Validation [4]. Dieses Kürzel steht für eine qualitätsgetriebene Entwicklung, die alle Phasen des Softwareentwicklungszyklus umfasst. Insbesondere setzt man auf folgende Trumpfkarten:

- Spezifikations- und Konfigurationsmanagement,
- Anforderungsüberwachung und Codeinspektion,
- Codierungsstandards und Softwaremetriken,
- statische und dynamische Analyse,
- automatisiertes Testen und Debugging,
- Simulation,
- formale Verifikation und Modellprüfung.

Und damit sind wir beim zweiten Teil der heutigen Vorlesung, nämlich:

2 Formale Methoden im Softwaredesign

Was verstehen wir unter „formalen Methoden“? Im Allgemeinen sind das all die Methoden, bei denen zwischen der informellen, verbalen Anforderungsdefinition (einer rosa Wolke) und dem ausführbaren Code, welcher diese Anforderungen realisiert (einer grauen Schachtel), eine weitere Ebene liegt, nämlich die Ebene der *formalen Beschreibung*. Den Weg von der Anforderungsdefinition zur formalen Beschreibung bezeichnet man als *Spezifikation*. Der Beweis, dass ein System die formalen Anforderungen erfüllt, heißt *Verifikation*. Aus einer formalen Beschreibung können *Tests* für das fertige System generiert wer-

den. Und wann bezeichnen wir eine Beschreibungssprache als „formal“? Wir fassen diesen Begriff sehr weit und verstehen darunter alle die Sprachen, die folgende Anforderungen erfüllen:

- Sie haben eine feste *Syntax*: Es lässt sich zweifelsfrei feststellen, ob eine Beschreibung sprachlich korrekt ist oder nicht.
- Sie haben eine klare *Semantik*, die keinen Spielraum für subjektive Interpretationen der Bedeutung einer Beschreibung lässt. Oft ist die Semantik mit mathematischen Mitteln formuliert.
- Sie haben irgendeine Art von *Deduktionssystem*, welches es erlaubt, aus einer Beschreibung Ergebnisse zu berechnen, sie auszuführen, umzuformen, Aussagen darüber zu beweisen oder ähnliches.

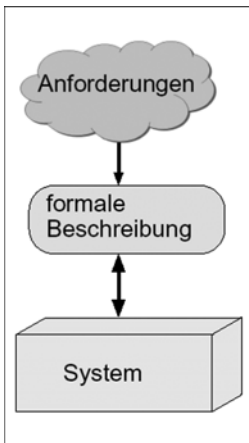


Abb. 5:
Formale Methoden in der Software-Entwicklung

Eine formale Methode besteht aus der formalen Beschreibungssprache sowie Werkzeugen zur Bearbeitung von Beschreibungen zur Anwendung des Deduktionsapparates. Die Stärke einer formalen Methode bemisst sich oft nach der praktischen Anwendbarkeit des Deduktionssystems auf „sehr große“ Beschreibungen (Abb. 5).

Natürlich sind die Grenzen des Begriffs fließend; beispielsweise ist es fraglich, ob die *Unified Modelling Language* (UML) eine formale Semantik hat (für UML-Statecharts gibt es Dutzende von Vorschlägen für eine Semantik). Am anderen Ende des Spektrums könnte man auch jede herkömmliche Programmiersprache wie z. B. C als formale Methode bezeichnen, wenn man als Deduktionssystem die Ausführung durch eine virtuelle Maschine nimmt. Allerdings lassen sich auf diese Weise nur sehr eingeschränkte Aussagen über ein Programm „beweisen“, nämlich das Ausgabeverhalten bei einer bestimmten vorgegebenen Eingabe.

Typische formale Methoden, wie sie in der Industrie heute eingesetzt werden, sind StateMate Statecharts, die „Object Constraint Language“ (OCL) der UML, die „Specification and Description Language“ (SDL) in der Telekommunikation, und andere. Ich selbst habe bislang vornehmlich mit Petrinetzen und Temporallogik gearbeitet. Weitere interessante Sprachen aus dem akademischen Bereich, deren Einfluss für praktische Anwendungen vermutlich noch zunehmen wird, sind Abstract State Machines (ASMs) und die verschiedenen algebraischen Spezifikationsprachen wie z.B. LOTOS, CSP und CASL. Die Liste ließe sich noch sehr weit fortsetzen; ein Überblick über gängige formale Methoden ist in [5] zu finden.

Als Beispiel für den Einsatz formaler Methoden in der Raumfahrt möchte ich ein fehlertolerantes Rechensystem vorstellen, an dessen Entwicklung wir beteiligt waren und welches sich der-

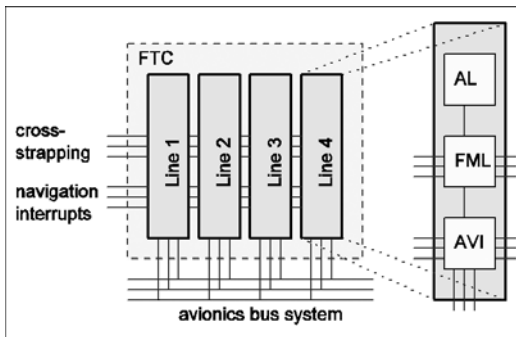
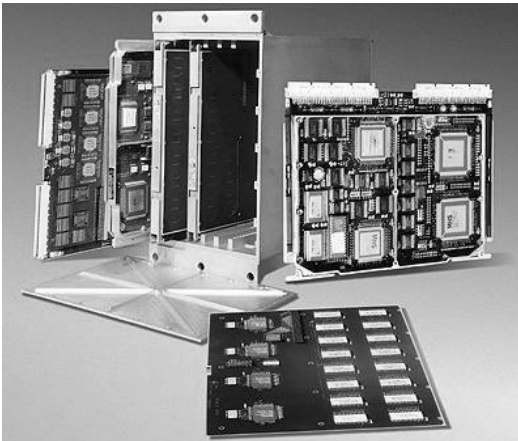


Abb. 6:
Interne Struktur des
DMS-R

zeit im Einsatz im Weltall befindet (Abb. 6). Es handelt sich um das DMS-R, das „Data Management System for the Russian Service Module“. Dieses System besteht aus zwei fehlertoleranten Computern (FTCs) und ist das „zentrale Gehirn“ der internationalen Raumstation: In der Konstruktionsphase der Station wird es für den korrekten Zusammenbau der Module verwendet, danach dient es als zentrales Datenverwaltungssystem für alle wissenschaftlichen Experimente an Bord, hält die Station korrekt in der Bahn, steuert den europäischen Roboterarm und navigiert und überwacht das Andocken der Transport- und Versorgungs-

fahrzeuge. Das System wurde von der Astrium AG in Bremen entwickelt und mit einem der ersten Transporte im Juli 2000 in den Orbit gebracht. Seitdem funktioniert es problemlos [6]. Es ist 4-fach redundant ausgelegt, wobei sich auf jeder Platine 3 Prozessoren befinden (Abb. 7).

Das „Application Layer“ ist zuständig für die Berechnungen: Hier werden die Programme ausgeführt, die die eigentliche Nutzlast des Systems bilden. Das „Fault Management Layer“ verbindet die einzelnen Repliken untereinander und sorgt für die Desintegration, den Neustart und die Reintegration fehlerhafter Teile. Das „Avionics Layer“ organisiert und kontrolliert den Zugriff des Systems auf den Datenbus der Raumstation: Es zerlegt ausgehende Nachrichten in einzelne Bestandteile („box cars“) und setzt eingehende Nachrichten aus den passenden Teilen wieder zusammen.



*Abb. 7:
DMS-R, der zentrale
Computer in der inter-
nationalen Raum-
station (ISS)
(© ESA)*

Beim Entwurf des DMS-R ergaben sich für Astrium folgende Fragen: Ist das Fehlertoleranzkonzept tragfähig? Kann es zwischen den einzelnen Prozessen zu einer Verklemmung kommen? Können wir für das Lastverhalten auf dem Bus einen gewissen Mindestdurchsatz garantieren? Zur Beantwortung dieser Fragen haben wir zunächst ein Modell der Software-Struktur des Avionics Layer entworfen. Ein kritischer Punkt war dabei der zentrale

Scheduler: da alle Prozesse mehrfach vorkommen können, war hier ein potentieller Flaschenhals zu erwarten. Ein weiterer kritischer Punkt bestand aus dem internen Puffer, der durch einen so genannten „dual ported RAM“ realisiert war, einen Speicher, der von zwei Seiten gleichzeitig gelesen und geschrieben werden kann (Abb. 8). Falls dieser Puffer überlaufen sollte, werden Daten an die höhergelegenen Schichten zurückgewiesen, was die gesamte Übertragungsrate massiv senken würde. Das Scheduling selbst ist abhängig von Höchst- und Mindestlaufzeiten der einzelnen Prozesse, die wiederum durch die feste Länge der einzelnen „box cars“ auf dem Bus mit den fest vorgegebenen Takt-raten bestimmt wird.

Es handelt sich also hierbei um ein komplexes *Echtzeitsystem*. Das sind solche Systeme, bei denen nicht nur kausale, sondern auch quantitative Aspekte des Zeitverhaltens für die Korrektheit

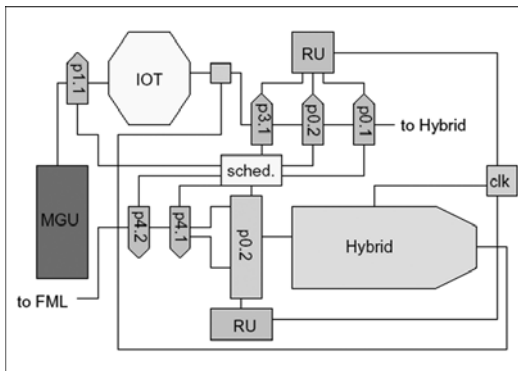


Abb. 8:
Modellierung des
Avionics Layer des
DMS-R

eine Rolle spielen. Was bedeutet das? Ein Echtzeitsystem ist gekennzeichnet dadurch, dass seine Spezifikation von einer externen Uhr abhängt, die nicht zum System gehört. Um Zeit „objektiv“ messen zu können, ist ein Bezugspunkt außerhalb des messenden Systems erforderlich. Ein Echtzeitsystem erfüllt sozusagen einen Kontrakt mit seiner Umgebung, der in der Spezifikation beschrieben ist. Dieser nimmt auf „absolute“ Zeitdauern und Zeitpunkte Bezug, die sich nach einer Uhr außerhalb des Systems richten. Wir sprechen in diesem Zusammenhang auch

von einer „globalen Uhr“, von der die Korrektheit des Systems abhängt. Um einem weit verbreiteten Missverständnis vorzubeugen: „Hard real time“ bedeutet nicht, dass das System „so schnell wie möglich“ oder „in extrem kurzer Zeit“ reagiert, sondern lediglich, dass das Zeitverhalten skalierbar und in Bezug auf eine externe Zeitmessung vorhersagbar ist.

Oftmals ist es besser, in der Spezifikation Unter- und Obergrenzen für die Dauer von Ereignissen statt des exakten Zeitpunktes ihres Eintreffens festzulegen. Also etwa „zwischen 11:59 und 12:01“ statt „um 12:00“, wobei als Referenz die Zeiger des Big Ben oder die Atomuhren der Physikalisch-Technischen-Bundesanstalt in Braunschweig [7] gelten können. Abgesehen von prinzipiellen Überlegungen zur Struktur der Zeit und zur Genauigkeit unserer Zeitbeobachtung hat dies ganz praktische Gründe. Betrachten wir als Beispiel eine Ampelsteuerung mit Lichtzeichenanlage und Überquerungsanforderungsknopf. Eine qualitative Anforderung wäre zum Beispiel:

„Wenn der Knopf gedrückt wird, geht schließlich das grüne Licht an.“

Eine quantitative Anforderung hingegen wäre etwa:

„Wenn der Knopf gedrückt wird, dauert die Rotphase noch mindestens 10 Sekunden.“

(Diese Forderung würde garantieren, dass sich nähernde Kraftfahrzeuge noch durchfahren können, ohne abrupt bremsen zu müssen), sowie zusätzlich:

„Wenn der Knopf gedrückt wird, dauert die Rotphase noch höchstens 60 Sekunden.“

(Diese Forderung ist notwendig, damit Fußgänger nicht ungeduldig werden und bei Rot über die Straße gehen.) Bereits in diesem einfachen Beispiel sehen wir die Notwendigkeit der Verwendung von Unter- und Obergrenzen; eine Anforderung der Art:

„Wenn der Knopf gedrückt wird, dauert die Rotphase noch genau 30 Sekunden.“

wäre wesentlich schwieriger bzw. überhaupt nicht realisierbar. Ein anderes Beispiel sind Eigenschaften für Realzeitprotokolle wie im Beispiel des DMS-R. Eine Anforderung hier war die Garantie eines gewissen Minstdurchsatzes, etwa, dass die Übertragungsrate mindestens 10 Mbit/s beträgt. Dies kann man ausdrücken durch die Eigenschaft:

„Der Abstand zwischen je zwei empfangenen Bit beträgt höchstens 0,1 Millisekunden.“

Andererseits gibt es für kollisionsbehaftete Protokolle wie etwa im Ethernet Eigenschaften, die gewisse Mindestabstände zwischen den Nachrichten fordern, etwa:

„Nach einer Kollision muss vor der Neuübertragung mindestens 0,01 Millisekunden lang gewartet werden.“

Auch hier haben wir also Anforderungen, die sich durch Mindest- und Höchstzeiten beschreiben lassen. Generell ist diese Art von Eigenschaften typisch für Realzeit-Spezifikationen.

Wie werden Realzeitsysteme heutzutage konstruiert? Normalerweise realisiert man die Software nach der Maxime „so günstig wie möglich, so effizient wie nötig“. Das bedeutet, man legt während des Entwurfs zunächst keinen Wert auf die zeitlichen Anforderungen. Später, nach der Integration von Hard- und Software, misst man dann das Zeitverhalten am fertigen System. Falls die geforderten Mindestzeiten unterschritten werden, d.h., das System ist zu schnell, kann man sich mit zusätzlich eingebauten Warteschleifen behelfen. Die Dauer der Warteschleife wird als Parameter definiert, der mit weiteren Messungen eingestellt wird. Wenn das System zu langsam ist, d.h. wenn Höchstzeiten überschritten werden, versucht man, durch Optimierungen im Code die Ausführungsgeschwindigkeit zu steigern. Falls das nicht hilft, setzt man schnellere und teurere

Hardware ein, zum Beispiel Spezialprozessoren oder Parallelverarbeitung.

Dieses „trial-and-error“-Vorgehen bereitet aber erhebliche Probleme. Als erstes ist zu bemerken, dass die gemessenen Ausführungszeiten immer gewissen Abweichungen unterworfen sind. Ganz abgesehen von der Tatsache, dass jede Messung das Messergebnis beeinflusst, ist es unter Laborbedingungen zumeist nicht möglich, die spätere Produktivumgebung vollständig zu simulieren. Daher sind die gemessenen Zeiten mit einer gewissen Unsicherheit versehen. Darüber hinaus jedoch, und das ist fast noch schwerwiegender, werden durch diese Vorgehensweise zusätzliche, nicht notwendige Abhängigkeiten zwischen der auszuführenden Software und der ausführenden Hardware geschaffen: Die Messungen gelten nur für die spezielle Konfiguration, mit der sie durchgeführt wurden. Bei jedem Wechsel der Technologie ist der gesamte Prozess erneut zu durchlaufen, was oftmals einem kompletten Reengineering gleich kommt.

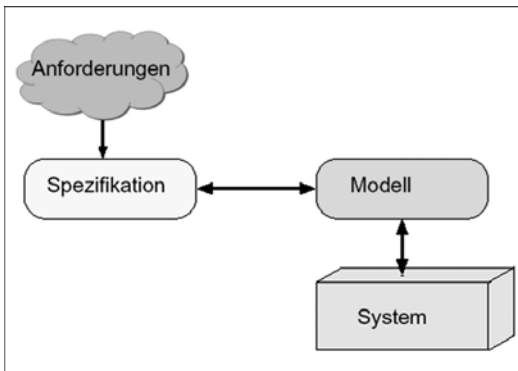


Abb. 9:
Paradigma des „model
checking“

Wir beschäftigen uns daher mit modellbasierter Softwareentwicklung: Das sind alle diejenigen formalen Methoden, bei denen die formale Beschreibungsebene aus zwei Teilen besteht: aus der Spezifikation, die eine exakte Beschreibung der geforderten Eigenschaften des Systems ist, und dem Modell, welches die Funktionsweise des Systems beschreibt (Abb. 9).

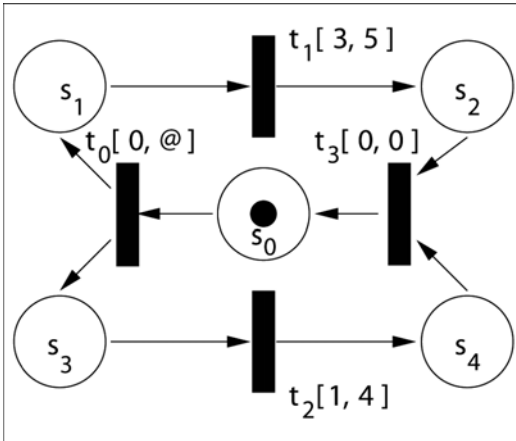


Abb. 10:
Ein Zeit-Petrinetz
Beispiel

Zur Modellierung verwenden wir nachfolgend die so genannten „Zeit-Petrinetze“ von Berthomieu und Diaz [8]. Dieser Formalismus ist eine Erweiterung der „klassischen“ Petri-Netze um Realzeitkonstrukte, mit denen Mindest- und Höchstgrenzen im Zeitverhalten ausgedrückt werden können. Ein Petrinetz ist ein bipartiter Graph aus Stellen und Transitionen, die mit Pfeilen verbunden sind. Die Stellen repräsentieren Zustände, die Transitionen Ereignisse im System. Auf jeder Stelle kann sich eine Marke befinden, die einen Prozess repräsentiert. Eine Transition ist schaltbar, sobald alle Eingangsstellen besetzt und alle Ausgangsstellen frei sind. Das Schalten einer Transition modelliert einen Ausführungsschritt des Systems. In Zeit-Petrinetzen hat jede Transition eine eigene Uhr; diese beginnt mit Null zu laufen, sobald die Transition schaltbar wird. Zu jeder Transition gibt es eine früheste und späteste Schaltzeit (eine Zahl zwischen Null und unendlich). Eine schaltbare Transition kann erst nach Ablauf der frühesten, muss aber nach Ablauf der spätesten Schaltzeit schalten, wenn sie solange schaltbar bleibt. In Abb. 10 ist ein Beispiel für ein Zeit-Petrinetz angegeben.

Ein Zustand des Netzes ist gegeben durch die Belegung der Stellen mit Marken und die Stellung aller Uhren der Transitionen. Eine Schaltfolge ist eine Folge von solchen Zuständen; statt der Stellung aller Uhren reicht es, die abgelaufene Gesamtzeit anzu-

geben. Für das in Abb. 10 (S. 21) angegebene Zeit-Petrinetz wäre eine mögliche Schaltsequenz die folgende.

$((\{s_0\}, 0) (\{s_1, s_3\}, 10) (\{s_2, s_3\}, 13) (\{s_2, s_4\}, 14) (\{s_0\}, 14) \dots)$

Als ein realistischeres Beispiel geben wir die Modellierung einer „Message Generation Unit“ im DMS-R mit Zeit-Petrinetzen an. Diese Einheit dient in unserer Simulation als Quelle für Nachrichten, die auf den Bus gelegt werden sollen. In Abb. 11 verwenden wir zusätzlich aus Effizienzgründen inhibitorische Kanten. Die Zeiten sind variabel gehalten und separat gespeichert.

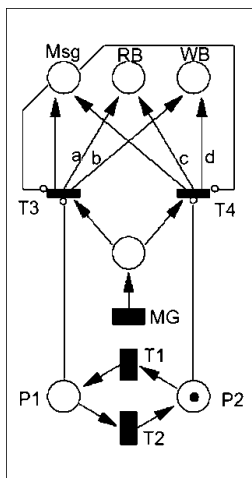


Abb. 11:
Modellierung der
Nachrichtenerzeugung
im DMS-R

In diesem Beispiel ist die gewünschte Eigenschaft folgende: Es werden in rascher Abfolge „kurze“ und „lange“ Nachrichten produziert und auf den Stellen RB („reading boxcars“) und WB („writing boxcars“) bereit gestellt. Eine Frage, der wir uns später zuwenden wollen, ist die, wie solche Anforderungen formal sauber aufgeschrieben werden können.

In dem geschilderten Beispiel waren Zeit-Petrinetze nicht der einzige Formalismus, den wir zur Modellierung verwendet haben. Ein anderes Modell wurde aus dem OCCAM-Quellcode in der Prozessalgebra „Timed CSP“ [9] extrahiert. Dies ist ein alternativer Formalismus zur Beschreibung von Modellen; prinzipiell könnte man Zeit-Petrinetze nach Timed CSP übersetzen und umgekehrt. Timed CSP ist eine Erweiterung der Sprache CSP (Communicating Sequential Processes) von C. A. R. Hoare [10], aus der auch die Programmiersprache OCCAM abgeleitet wurde. Das Avionics Layers des DMS-R wurde in OCCAM implementiert, da die verwendete Transputer-Hardware diese Sprache besonders gut unterstützt.

Das Bestechende an formalen Notationen ist ihre verblüffend einfache Syntax. So lässt sich etwa die komplette Syntax der Sprache Timed CSP in nur zwei Zeilen notieren:

$$\alpha ::= STOP \mid SKIP \mid q \mid (a \longrightarrow \alpha) \mid (\alpha \parallel \alpha) \mid (\alpha \sqcap \alpha) \mid (\alpha; \alpha) \\ \mid (\alpha \underset{\Delta}{\parallel} \alpha) \mid (\alpha \overset{\dagger}{\parallel} \alpha) \mid \nu q \alpha \mid \alpha\{q_1 := q_2\}$$

Auch die exakte Semantik für Terme dieser Sprache lässt sich erstaunlich einfach angeben. Im Wesentlichen reicht es, für jedes syntaktische Konstrukt eine Regel anzugeben, welche die Wirkung beschreibt:

- $((SKIP, t), \checkmark, (STOP, t)) \in \Delta$.
- Wenn $a \in \Sigma$, dann $((a \longrightarrow \alpha), t), a, (\alpha, t) \in \Delta$.
- Wenn $((\alpha_1, t), a, (\alpha'_1, t')) \in \Delta$, dann $((\alpha_1; \alpha_2), t), a, ((\alpha'_1; \alpha_2), t') \in \Delta$.
- $((\alpha_1 \overset{\dagger}{\parallel} \alpha_2), s), \tau, (\alpha_2, s + t) \in \Delta$.
- ...

Hier ist ein Beispielfür einen CSP-Ausdruck, das aus einer anderen Raumfahrtanwendung stammt (der Energie- und Thermalkontrolle eines Forschungssatelliten). Beachten Sie, dass auch hier in der tatsächlichen Implementierung (in der Sprache des Werkzeugs FDR, [11]) einige zusätzliche Operatoren vorkommen, die in der „reinen“ Syntax nicht enthalten sind.

```
SPEC = ( SWITCHDEV [| | Tau_nextTC |] TCTIM ) ||| TIMCHK
SWITCHDEV = Tau_nextTC -> (
  (Com_PYRO_PWR_DEV_ON -> setTimSwt ->
   Swt_BS_ON_MAIN_ON -> Swt_PYRO_PRE_MAIN_ON -> Swt_PYRO_PWR_MAIN_ON ->
   resTimSwt -> SWITCHDEV)
|^| (Com_PYRO_PWR_DEV_OFF -> setTimSwt ->
   Swt_PYRO_PWR_MAIN_OFF -> resTimSwt -> SWITCHDEV)
|^| ... )

TIMCHK = elaTimSwt -> errorSwitchTimer -> TIMCHK
TCTIM = Tau_nextTC -> setTimTick -> elaTimTick -> TCTIM
```

Dieses „Codesegment“ zeigt, dass in der tatsächlichen Praxis die Modellierungssprache genau wie eine Programmiersprache verwendet wird. Der konkrete Ausschnitt realisiert folgende Anforderung an die Schaltfunktionalität, so wie sie im Code aus dem Pflichtenheft umgesetzt wurde:

At any given moment, it is possible to send telecommands for turning any device on or off. All switching operations necessary to activate or deactivate this device must be performed within a given time constant.

Wie man sieht, ist die Modellierung dieser Anforderung sehr nahe an der tatsächlichen Implementierung in einem Programm. Zur formalen Spezifikation ist es besser, Sprachen zu verwenden, die sich mehr an der verbalen Beschreibung in natürlicher Sprache orientieren.

Bereits seit Aristoteles ist es der Gegenstand der Logik, Argumentationen, die in natürlicher Sprache ausgedrückt sind, zu formalisieren. Im zwanzigsten Jahrhundert wurden viele verschiedene logische Sprachen entwickelt, zunächst vor allem, um Axiome und Regeln der Mathematik exakt beschreiben zu können, später auch für Sachverhalte und Schlussweisen in anderen Gebieten. Seit Beginn des Computerzeitalters benutzt man logische Formeln auch dazu, die Anforderungen an ein Programm „formal“ festzulegen. Wie bereits oben erläutert, besteht eine logische Spezifikation aus einer Menge von Formeln, die das Sollverhalten des Programms unzweideutig beschreiben. Der Unterschied zwischen Modellierung und Spezifikation besteht darin, dass ein Modell eher zustandsorientiert ist und das operationale Verhalten eines Systems reflektiert, während logische Formeln eher ereignisorientiert sind und die erwünschten Eigenschaften des Systems widerspiegeln. Natürlich ist dieser Unterschied nur ein gradueller: In vielen Fällen kann man ein Modell automatisch in eine Menge von Formeln transformieren und umgekehrt.

Wir benutzen zur Spezifikation meist temporale Echtzeit-Logiken. Hier gibt es viele Varianten; „jeder Logiker, der etwas auf sich hält, hat einmal seine eigene Logik definiert“. Natürlich stehen auch die Informatiker dem nicht nach: In Zusammenarbeit mit dem Tokyo Institute of Technology haben wir eine Sprache **TNL** („temporale Netzlogik“) entwickelt, die speziell auf Zeit-Petrinetze zugeschnitten ist. Für jede Stelle im Netz enthält diese

Logik eine atomare Aussage mit der Bedeutung „die Stelle ist markiert“. Darauf aufbauend wird die Sprache induktiv definiert:

- Jede atomare Aussage ist eine Formel.
- \perp („falsum“) ist eine Formel.
- Wenn φ_1 und φ_2 Formeln sind, so auch $(\varphi_1 \rightarrow \varphi_2)$.
- Wenn φ_1 und φ_2 Formeln sind, so auch $\Box \varphi_1$ („always“) und $(\varphi_1 U \varphi_2)$ („until“).
- Wenn p und q Stellen im Netz sind und c eine rationale Zahl ist, dann ist $p^*-q^* \leq c$ eine Formel. Dabei steht der Stern für eines der beiden Symbole \circ oder \bullet .

Diese Definition erlaubt es, beliebig komplexe Formeln zusammensetzen; zum Beispiel wäre

$$((p^\bullet - q^\circ \leq 4 \rightarrow \perp) U (q^\circ - p^\bullet \leq \rightarrow p) U \Box q)$$

eine syntaktisch zulässige Formel. Die in der Praxis vorkommenden Formeln sind allerdings meist wesentlich einfacher. Die Semantik wird gemäß dem induktiven Aufbau der Formeln definiert. Wir betrachten dazu die Schaltfolgen, die sich durch die Ausführung des zu Grunde liegenden Petri-Netzes ergeben. Für jede Formel und Schaltsequenz legen wir fest, ob die Formel erfüllt ist oder nicht. Für einen bestimmten Zustand in der Folge ist der Wert (wahr oder falsch) jeder atomaren Aussage durch die Markierung des Netzes festgelegt. Die Formel \perp ist niemals erfüllt, die Formel $(\varphi_1 \rightarrow \varphi_2)$ ist erfüllt, falls φ_1 erfüllt oder φ_2 nicht erfüllt ist. $\Box \varphi_1$ ist in einem Zustand erfüllt, falls φ_1 in allen nachfolgenden Zuständen der Folge erfüllt ist, und $(\varphi_1 U \varphi_2)$ ist erfüllt, wenn φ_1 in allen Zuständen bis hin zum nächsten φ_2 erfüllt ist. Intuitiv bedeutet p^\bullet den Zeitpunkt, zu dem p zum letzten Mal erfüllt wurde, und p° ist die Zeit, zu der p zum letzten Mal nicht erfüllt wurde. Die Ungleichung $p^*-q^* \leq c$ ist erfüllt, wenn sie mit dieser Interpretation der Variablen gültig ist. Eine Formel ist in einer gesamten Schaltfolge erfüllt, wenn sie in deren Anfangszustand erfüllt ist.

Hier ist ein realistisches Beispiel für die Spezifikation von Systemeigenschaften. Eine Eigenschaft im oben genannten Satellitencontroller lautete „Wann immer Signal p gesetzt wird, wird es innerhalb von 5 Zeiteinheiten wieder gelöscht“. Bildlich kann man diese Anforderung etwa wie in Abb. 12 darstellen:

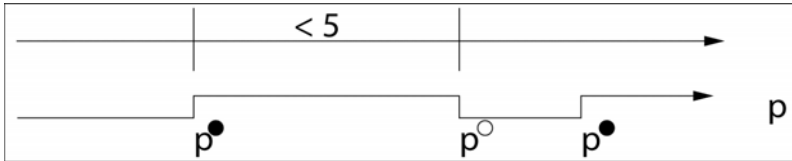


Abb. 12:
Ein möglicher Zeitlauf

Als Formel der TNL lässt sich der Sachverhalt ganz einfach durch $\square(p^\circ - p^\bullet \leq 5)$ spezifizieren.

Als zweites Beispiel soll uns die Beschreibung für einen regelmäßigen Weckruf-Mechanismus in einem Übertragungsprotokoll dienen. Im Pflichtenheft findet sich folgende Passage:

„Each time the timer expires a poll is transmitted and the timer is restarted.“

Diese sehr operationale Beschreibung ist von jemandem aufgeschrieben worden, der bereits eine konkrete Realisierung vor Augen hatte: Mit Sicherheit stammt dieser Satz von einem Menschen mit Programmiererfahrung! Es ist sehr einfach, ein Zeit-Petrinetz als Modell anzugeben, welches diese Beschreibung realisiert. Wir könnten zwar auch eine logische Formulierung dafür angeben, jedoch wäre diese vermutlich sehr kompliziert und unnatürlich. Das Problem ist, dass sich der Satz auf das „wie“ statt auf das „was“ konzentriert. Die erwünschte Systemeigenschaft, um die es eigentlich geht, lässt sich vermutlich besser beschreiben durch die Forderung

„Alle k Zeiteinheiten wird ein *poll* gesendet“.

Diese Anforderung ist wohl sofort und unmittelbar einleuchtend! Sie lässt sich sofort als **TNL**-Formel aufschreiben, nämlich als $\Box(|poll^\bullet - poll^\circ| = k)$. (Wir verwenden hier die üblichen abkürzenden Notationen, etwa $x = y$ für $(x \leq y \wedge y \leq x)$ usw.) Wenn man allerdings einen Moment über diese Forderung nachdenkt, so merkt man, dass sie in der vollen Strenge nicht realisierbar ist: die Zeitmessung sowohl von Menschen als auch von Maschinen ist immer mit einer gewissen Messungenauigkeit behaftet. Daher ist es vermutlich sinnvoller, obige Forderung wie folgt abzuschwächen:

„Circa alle k Zeiteinheiten wird ein *poll* gesendet“.

In **TNL** lässt sich das durch $\Box(k - eps \leq |poll^\bullet - poll^\circ| \leq k + eps)$ formalisieren (vgl. Abb. 13).

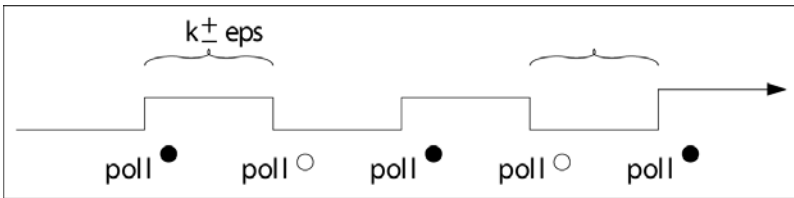


Abb. 13:
Anforderung für den Zeittakt

3 Modellprüfung, Konformanzanalyse, Testautomatisierung

Wir haben vorhin eine *formale Methode* dadurch charakterisiert, dass sie eine feste Syntax und klare Semantik hat, sowie dadurch, dass ein Deduktionssystem existiert, mit dem semantische Aussagen über die syntaktischen Beschreibungen bewiesen werden können. Als Modellierungsfomalismus haben wir Zeit-Petrinetze betrachtet, wobei die Semantik eines Netzes die Menge aller seiner Schaltfolgen (Abläufe) ist. Zur Spezifikation von Systemeigenschaften haben wir die temporale Netzlogik **TNL** definiert, wobei die Semantik einer Formel die Menge aller der Schaltfol-

gen ist, in denen sie erfüllt ist. Dieser Ansatz erlaubt es uns nun, die informelle Frage „Erfüllt das System die gestellten Anforderungen?“ zurückzuführen auf die formale Frage „Erfüllt das Modell die Spezifikation?“. Mit anderen Worten, wir sind an folgender Aufgabenstellung interessiert: „Gegeben ein Netz N und eine Formel φ ; ist die Formel in jedem Ablauf des Netzes erfüllt?“ Deduktionssysteme zur Lösung dieser Aufgabe nennt man Modellprüfer („Model Checker“) [12]. Wir haben bereits vor einiger Zeit für die oben genannten Formalismen einen Modellprüfungsalgorithmus entwickelt, der auf der so genannten Partialordnungsmethode basiert und das Problem der Echtzeit-Modellprüfung auf bekannte Standard-Modellprüfungsverfahren reduziert. Ohne auf die genauen Details eingehen zu wollen, sei hier nur kurz erwähnt, dass die Kernidee in diesem Algorithmus darin besteht, einen endlichen *Regionengraphen* zu konstruieren, wobei die Pfade durch diesen Graphen exakt die Abläufe des Netzes repräsentieren, und jeder Knoten des Graphen den Wahrheitswert jeder Ungleichung in der gegebenen Formel bestimmt. Äquivalenzklassen von Zuständen im Netz werden durch Mengen von Ungleichungen im Regionengraphen repräsentiert, so genannte „clock zones“. Um den Regionengraphen endlich zu halten, verwenden wir unter anderem zwei Eigenschaften:

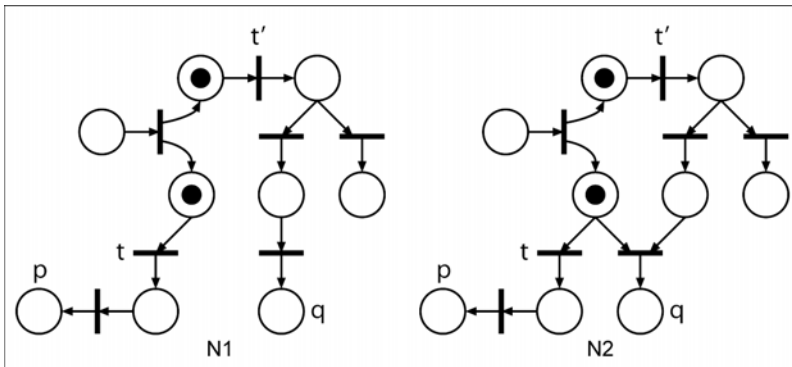
- überflüssige Variablen (die in der Formel nicht vorkommen) können aus der Ungleichungsmenge ohne Informationsverlust eliminiert werden, und
- Differenzen zwischen Variablen können ab einer bestimmten Größe als unendlich betrachtet werden.

Im Anschluss an unseren Vorschlag wurden von anderen Forschergruppen verschiedene alternative Repräsentationsformen untersucht und in ähnlichen Algorithmen verwendet, nämlich so genannte „difference bound matrices“, „clock regions“, „difference decision diagrams“, und andere, siehe zum Beispiel Arbeiten in dem inzwischen jährlichen Workshop RT-TOOLS [13]. Trotz kompakter Repräsentation sind jedoch die Regionengraphen in realen Anwendungsbeispielen „sehr groß“! Im Allgemeinen verhalten sich Modellprüfungsalgorithmen nicht lokal

Durch Äquivalenzklassenbildung auf stotteräquivalenten Modellen kann die Durchschnittskomplexität der Modellprüfung gesenkt werden [14]. Diese Techniken sind unter dem Namen Partialordnungsreduktionen bekannt geworden; die entscheidende Idee ist dabei, dass die Reihenfolge von Transitionen, die voneinander unabhängig ausgeführt werden, keine Rolle für das Gesamtergebnis spielt und die Ausführungsreihenfolge daher vertauscht werden kann, ohne dass das Ergebnis sich dadurch ändert. Bei Realzeitsystemen können wir dabei so weit gehen, dass wir ein „späteres“ mit einem „früheren“ Ereignis vertauschen und sich die kausale und zeitliche Reihenfolge umdreht. Angenommen, in einer Region seien die Transitionen t und t' schaltbar. Falls für jede mit t' beginnende Sequenz eine mit t beginnende äquivalente Sequenz existiert, so nennen wir t unabhängig von t' . Wir können dann bei der Konstruktion des Regionengraphen den gesamten von t' ausgehenden Teilgraphen vernachlässigen, da er für die Korrektheit des Ergebnisses keine Rolle spielt. Bildlich kann man sich das etwa so vorstellen, wie es in Abb. 14 (S. 29) dargestellt ist: Der gesamte linke Teil dieses Graphen braucht nicht berücksichtigt zu werden, da die dort enthaltenen Zustände das Gesamtergebnis nicht verändern.

In Abb. 15 sind zwei Netze, bei deren Analyse diese Situation entstehen könnte: Wenn in der Formel jetzt die Stellen p und q

Abb. 15:
Zwei Beispielnetze



erwähnt werden, dann ist t unabhängig von t' in N1, aber nicht in N2. Im ersten Fall spielt es keine Rolle, ob wir zuerst t und dann t' schalten oder umgekehrt: bei beiden Reihenfolgen werden alle Aussagen über p und q unverändert sein. Im zweiten Fall ist dies nicht so: Wenn in N2 zuerst t schaltet, ist q nicht erreichbar; wenn zuerst t' schaltet, ist es möglich, dass p nie erreicht wird. Wir sehen also, dass es sehr stark von der syntaktischen Struktur des Netzes abhängt, ob die Schaltreihenfolge zweier Transitionen wichtig für das Ergebnis ist oder nicht. Im Falle von Zeit-Petrinetzen hängt es darüber hinaus auch von den frühesten und spätesten Schaltzeiten, die mit den Transitionen verbunden sind, ab. Daher haben wir eine Relation definiert, aus der ersichtlich ist, ob zwei Transitionen eines Zeit-Petrinetzes voneinander unabhängig sind, und die für jeden gegebenen Zustand leicht berechnet werden kann.

Zur Reduktion des Regionengraphen wählen wir bei der Konstruktion in jedem Zustand irgendeine feuerbare Transition aus und bestimmen dann die Menge von Transitionen, die von ihr nicht unabhängig sind. Statt *aller* schaltbaren Transitionen genügt es dann, die Transitionen in dieser Menge zu betrachten, um eine Widerlegungssequenz für die gegebene Formel zu erhalten. Durch diese Vorgehensweise ist es uns in einigen Fällen gelungen, eine drastische Verkleinerung des Regionengraphen zu erreichen: während der vollständige Graph ein exponentielles Wachstum zeigte, wuchs der reduzierte Graph nur linear in der Größe des Netzes. Dieses Verhalten ist typisch für Systeme mit einem hohen Verteilungsgrad und vielen unabhängigen Aktivitäten.

Eine Reihe von Autoren haben diesen Ansatz danach verbessert und weiter verfeinert; in erster Linie sind hier zu nennen die Arbeiten von Bengtsson, Jonsson, Lilius, Yi [15], von Dams, Gerth, Knaack und Kuiper [16], sowie von Bieber und Fleischhack [17] und die Dissertation von Minea [18]. Neben unserer eigenen Implementierung in einem experimentellen Werkzeug ist inzwischen ein vergleichbares Modul in dem weit verbreiteten Modellprüfer UPPAAL [19] verfügbar. Aktuelle Forschungsthemen

in diesem Bereich sind die Erweiterung auf hybride Systeme sowie die Kombination mit symbolischen Repräsentationen des Zustandsraumes.

Trotz der vielen Verbesserungen, die es für die verschiedenen Modellprüfungsverfahren inzwischen gibt, bleibt ein prinzipielles Problem darin bestehen, dass es ein „flaches“ Verfahren ist. Damit meine ich, dass die Modelle ohne Berücksichtigung der Modulstruktur des tatsächlichen Systems abstrahiert werden, und dass die Interaktion zwischen den verschiedenen Modulen mit der Kommunikation innerhalb eines Moduls gleich behandelt wird. In einer hierarchischen Verifikation versucht man, die Modellbildung in mehreren Abstraktionsstufen zu gestalten. Die konkreteste Ebene ist dabei der Programmcode des tatsächlichen Systems, die abstrakteste die formale Beschreibung der Systemanforderungen. Dazwischen gibt es mehrere Zwischenstufen. Wenn es uns jetzt gelingt, von jeder Konkretisierung zu zeigen, dass sie in Bezug auf die jeweilige Abstraktion korrekt ist, dann können wir daraus folgern, dass das System die geforderten Eigenschaften erfüllt.

Was bedeutet es nun, dass eine Konkretisierung „korrekt“ in Bezug auf eine Abstraktion ist? In der Literatur wurden verschiedene Verfeinerungsbegriffe vorgeschlagen, von der Teilmengeneigenschaft der erzeugten Sprachen bis hin zur strukturellen Teilgrapheneigenschaft der zugehörigen Transitionssysteme. Wir verwenden eine Definition, die auf Dill [20] zurückgeht, und in der der Begriff „Korrektheit“ durch den Begriff „Konformanz“ ersetzt ist. Eine Konkretisierung M_C ist *konform* zu einer Abstraktion M_A , falls für jede Umgebung M_E gilt: wenn die Abstraktion in dieser Umgebung keine „Failures“ hat, so hat auch die Konkretisierung in derselben Umgebung keine Failures. (Ich verwende hier den Begriff „Failures“ statt „Fehler“, um anzudeuten, dass es sich hierbei um einen *terminus technicus* handelt, der später noch genauer definiert wird.) Formal:

$$\forall M_E: \text{Failures}(\{M_A, M_E\}) = \emptyset \rightarrow \text{Failures}(\{M_C, M_E\}) = \emptyset$$

Intuitiv gesprochen, darf die Konkretisierung in einer Umgebung nur dann ausfallen, falls die Abstraktion für diesen Fall nichts vorgesehen hat, keine weiteren Anforderungen beschreibt (s. Abb. 16).



Abb. 16:
Verfeinerung eines Modells in der Umgebung

Die Begriff „Failure“ in der Konformanzanalyse ist historisch mit demselben Begriff in der Theorie der Prozessalgebren verwandt. Failures entstehen vor allem in der Kommunikation zwischen verschiedenen Komponenten eines Gesamtsystems sowie in der Kommunikation zwischen einem System und seiner Umgebung. Wir haben verschiedene Varianten des Begriffs untersucht. Ein „Safety Failure“ liegt vor, wenn eine Komponente ein Ausgabesignal erzeugt (oder absendet), aber ein anderes Modul kann dieses nicht verarbeiten (oder empfangen). Intuitiv kann man sich vorstellen, dass das ankommende Signal beim Empfänger einen Kurzschluss oder eine andere ungewollte Reaktion auslöst. Im konkreten Fall entsteht ein „Safety Failure“ zwischen Umgebung und System, wenn die Umgebung ein Signal erzeugt, welches das System zum Absturz bringt, oder wenn das System ein unerwartetes Signal erzeugt, welches von der Umgebung nicht vorgesehen ist.

„Safety Failures“ sind in der Literatur gut etabliert, mit ihnen kann man Sicherheits- oder Invarianzeigenschaften von hierarchischen Systemen charakterisieren. Bei Realzeitsystemen gibt es eine weitere Klasse von Failures, die wir untersucht haben, nämlich die „Timing Failures“. Ein solcher Failure tritt auf, wenn in einem System ein Modul eine Eingabe von einem anderen Modul erwartet, diese aber nicht rechtzeitig zur Verfügung steht. Anders als für Safety Failures gibt es hier verschiedene

Möglichkeiten, diesen Begriff zu definieren, wobei sich jeweils subtile Unterschiede in der resultierenden Theorie ergeben. Intuitiv gesprochen bedeutet ein Timing Failure eine unzulässige Verzögerung in der Synchronisation: Der Modul wird in seiner Ausführung durch das fehlende Signal behindert, dadurch kommt es im Gesamtsystem zu Problemen.

Unabhängig von der genauen Definition des Begriffs „Failure“ können die Failures von Modulmengen wie bei der Modellprüfung durch die Konstruktion des Regionengraphen berechnet werden, wenn die einzelnen Module durch Zeit-Petrinetze gegeben sind. Und genau wie oben ist es auch für die Konformanzanalyse möglich, eine Partialordnungsreduktion während der Konstruktion des Regionengraphen anzuwenden!

Allerdings ist es das Ziel der Konformanzanalyse ja nicht, irgendwelche Failures zu berechnen, sondern zu zeigen, dass eine Konkretisierung konform zu einer Abstraktion ist. Dazu benötigen wir den Begriff des Spiegels. Das Spiegelbild eines Moduls entsteht, indem Ein- und Ausgaben vertauscht werden: Jede Eingabe eines Moduls ist eine Ausgabe des gespiegelten Moduls und umgekehrt (Abb. 17).



*Abb. 17:
Verifikation durch
Spiegelung*

Der Spiegelsatz [20] besagt jetzt, dass eine Konkretisierung konform zu einer Abstraktion ist, wenn die Komposition der Konkretisierung mit dem Spiegelbild der Abstraktion keine Failures hat. Das bedeutet, dass eine Konformanzanalyse durch die Konstruktion des Regionengraphen von Konkretisierung und gespiegelter Abstraktion erfolgen kann. Wenn während dieser Konstruktion keine Failures gefunden werden, dann ist die Konformanz bewiesen. Da Konformanz eine transitive Relation ist, kann dieser Schritt auf verschiedenen Abstraktionsgraden wiederholt werden; auf diese Weise ist es möglich, die Konformanz eines stark hierarchisch gebauten Systems von Zeit-Petrinetzen zu beweisen.

Unsere ursprüngliche Aufgabe war es jedoch, zu beweisen, dass eine Implementierung gewisse Anforderungen erfüllt. Unter gewissen Voraussetzungen können wir diesen Ansatz auf temporallogische Spezifikationen erweitern: Um zu zeigen, dass ein „flaches“ Modell eine Formel erfüllt, genügt es, eine Abstraktion zu finden, die die Formel erfüllt, und zu zeigen dass das Modell konform zu dieser Abstraktion ist. Dieser Ansatz hat sich in einigen von uns untersuchten Beispielen als außerordentlich erfolgreich erwiesen. Im Avionics Layer des DMS-R beispielsweise erfolgte eine Modellbildung in mehreren Stufen, wobei wir jede Stufe mit der darüber liegenden verglichen haben.

In der Zusammenarbeit mit industriellen Auftraggebern ist es oft nicht möglich, eine vollständige Verifikation eines Systems durchzuführen. Dies hat nicht nur technische, sondern auch organisatorische und juristische Gründe. Natürlich ist es problematisch, wenn bei einer Verifikation zum Beispiel von Dateninhalten einer Nachricht abstrahiert wird; es ist immerhin möglich, dass die Korrektheit genau von diesen Inhalten abhängt. Oft ist es jedoch nicht nur die Größe des Quellcodes, die eine Verifikation verhindert; manchmal liegt der Code für das System noch gar nicht in der endgültigen Fassung vor, oder er kann aus Lizenzgründen nicht offen gelegt werden. Die von mir eben geschilderten Ansätze lassen sich jedoch sehr weit auf spezifikationsbasiertes Testen von Echtzeitsystemen übertragen. Hierbei geht es um die Beziehung zwischen dem Modell und dem System; das System wird als „black box“ betrachtet, in das man nicht weiter hineinschauen kann. Die Ausgaben des Modells sind die Eingaben für das System und umgekehrt. Dies ist also genau dieselbe Vorgehensweise wie bei der Konformanzanalyse, wo die Ausgaben der Konkretisierung als Eingaben der gespiegelten Abstraktion (Ausgaben der Abstraktion) behandelt wurden und umgekehrt! Die Zusammenschaltung von Testsystem und zu testendem System entspricht der Komposition von Konkretisierung und gespiegelter Abstraktion. Das nachfolgende Bild (Abb. 18, S. 36) zeigt den generischen Aufbau beim modellbasierten Test.

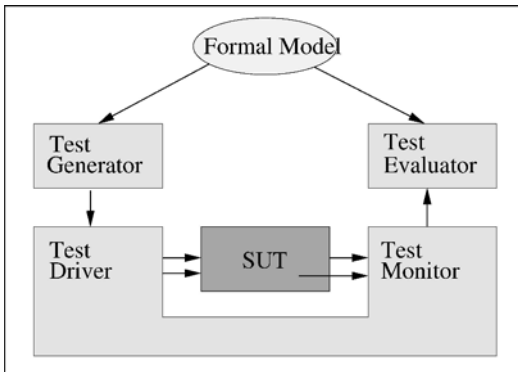


Abb. 18:
Modellbasierter Test

Hierbei ist „SUT“ eine gängige Abkürzung für „System Under Test“, das zu testende System. Die allgemeine Vorgehensweise bei der Durchführung der Tests ist wie folgt:

1. Schritt: Berechnung des Regionengraphen des Modells.
2. Schritt: Zusammenschaltung von Modell und System; statt der Berechnung der Failures erfolgt die Auswahl eines Zustandes der Region gemäß einer vorgegebenen Überdeckung.
3. Schritt: Auswertung der Testergebnisse.

Diese Vorgehensweise ist in verschiedenen modellbasierten Testwerkzeugen realisiert, zum Beispiel im RT-Tester des TZI-Spinoffs „Verified Systems International“ [21]. Wir haben dieses Werkzeug in verschiedenen Projekten eingesetzt, zum Beispiel in einer Testumgebung für den UMTS-Protokollstack in mobilen Endgeräten.

Die Konfiguration, die wir für das UMTS-Testbed aufgebaut haben, war so konfiguriert, dass das System auf verschiedenen Entwicklungsstufen mit denselben Testfällen konfrontiert werden konnte (Abb. 19, S. 37). Diese waren automatisch aus einer CSP-Modellierung der im vom 3GPP-Konsortium [22] festgelegten Eigenschaften generiert worden. Eine besondere Herausforderung war es, die Konfiguration so flexibel wie möglich zu halten, da während des Projektes sowohl die Anforderungen als auch die Implementierung ständig verändert wurden.

Eine aktuelle Aufgabe beim modellbasierten Testen besteht darin, die Testumgebung auf ein Netz von Testmaschinen zu verteilen, um auch mit hochgradig parallelen Testlingen und sehr engen zeitlichen Schranken zurechtzukommen.

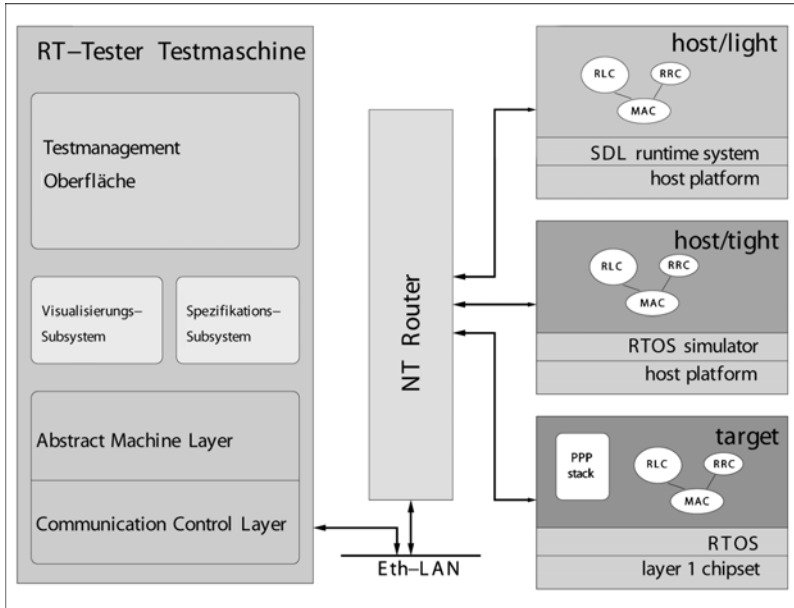


Abb. 19:
Aufbau der UMTS-Testumgebung

4 Forschungsperspektiven

Eine Antrittsvorlesung ist ein willkommener Anlass, um sich Gedanken über weitere gegenwärtige und künftige Herausforderungen und Forschungsgebiete zu machen. Ich möchte Ihnen zum Abschluss eine Liste der Themen präsentieren, die mich im Moment besonders interessieren.

4.1 Verbindung von Modellprüfung und modellbasiertem Testen

Obwohl ich mich bereits seit längerem mit diesen beiden Themen beschäftige, habe ich doch das Gefühl, dass die Zusammenhänge formal noch nicht genug untersucht sind. Ich habe Ihnen oben einige Beziehungen skizziert, die jedoch noch weiter ausbaufähig sind. Im Einzelnen sind das die folgenden Themen:

Welche Sprache überdeckt einen gegebenen Konformanzbegriff vollständig?

Vorhin haben wir gesehen, dass gewisse temporale Eigenschaften unter konformanten Verfeinerungen erhalten bleiben. Ist es möglich, eine temporale Logik zu definieren, mit der sich genau diese Eigenschaften formulieren lassen? Natürlich gehen die verschiedenen Varianten der Definition von Failures in diese Fragestellung mit ein. Kann man vielleicht sogar einen generischen Zusammenhang herstellen?

Testüberdeckungsstrategien für Realzeit

Verifikation und Test unterscheiden sich für Realzeitsysteme prinzipiell in der Hinsicht, dass bei der Verifikation mit beliebigen Intervallen der realen Zeitachse gerechnet werden kann, während beim Test immer nur einige ausgewählte Zeitpunkte ausgewählt werden können. Andererseits erlaubt es die Verifikation, die entscheidenden Zeitpunkte, an denen überhaupt etwas passieren kann, im Voraus zu berechnen. Kann man diese Eigenschaft ausnutzen, um geeignete Wartezeiten und Ereigniszeitpunkte beim Testen festzulegen?

Generierung von Testfällen aus Gegenbeispielen

Ein relativ neuer Trend in der Verbindung von Modellprüfung und Testen ist es, die von einem Modellprüfer generierten Gegenbeispiele als Testsequenzen für das Zielsystem zu benutzen. Eine offene Fragestellung hierbei ist die Auswahl von Eigenschaften, deren Gegenbeispiele dann zur Testsuite führen. Gibt es hier geeignete Heuristiken?

4.2 Beweismethodik

Einige eher grundlagenorientierte Fragen, die mich seit geraumer Zeit beschäftigen und vermutlich auch weiterhin beschäftigen werden, betreffen die Methodik des Modellprüfens mit temporaler Logik.

Automatisierte Abstraktionsmethoden, interaktives Beweisen

Ein wesentliches Problem für die Akzeptanz formaler Methoden in der Industrie ist es, dass die Erstellung von Modellen aus „fertigen“ Programmen einen zusätzlichen Arbeitsschritt bedeutet, der meist im Budget nicht vorgesehen ist. Wenn sich dieser Schritt schon nicht vollständig automatisieren lässt, kann man ihn dann wenigstens so weit wie möglich maschinell unterstützen? Wie muss ein interaktiver Beweiser konzipiert sein, der die Abstraktion von einem beweisbaren Modell aus ausführbarem Code unterstützt?

Expressivität temporaler Logiken

Obwohl viele Fragen der relativen Ausdrucksmächtigkeit temporaler Formeln und automatenorientierter Beschreibungsformen seit langem geklärt sind, gibt es doch noch einige interessante offene Probleme. Insbesondere ist oftmals die Komplexität der Übersetzungen von einem Formalismus in einen anderen ungeklärt, ganz zu schweigen von Heuristiken und Verfahren, die diese Übersetzungen praktikabel machen.

Modale und temporale Regelsysteme, offene Systeme

Gerade im Zusammenhang mit der Spezifikation offener Systeme, die in verschiedenen und in wechselnden Umgebungen eingesetzt werden können, taucht der Wunsch auf nach Erweiterungen der Möglichkeiten „klassischer“ temporaler Logiken wie LTL und CTL. Dabei interessiert mich die Frage, wie temporale Axiomensysteme für die Spezifikation eingesetzt werden können, und was die Möglichkeiten und Grenzen zweistufiger Ausdrucksmittel und alternierender Konzepte sind.

4.3 Spezifikationssprachen

Um die Akzeptanz formaler Methoden für reale Anwendungen zu fördern, ist es sicher nötig, die abstrakte Formelsprache auch für Leute verständlich zu machen, die nicht unbedingt eine Spezialausbildung in mathematischer Logik haben. Ich halte dies für eine der wichtigsten Aufgaben der Softwaretechnik in den nächsten Jahren und Jahrzehnten.

Automatisierte Umsetzung domänenspezifischer Formalismen

In den einzelnen ingenieurwissenschaftlichen Disziplinen haben sich jeweils spezielle Techniken und Notationen entwickelt, um Anforderungen aufschreiben zu können. Beispielsweise hat einmal ein Elektroingenieur versucht, mir einen Algorithmus zu erklären, indem er mir ein Schaltbild aufzeichnete. In der Verkehrstechnik arbeitet man viel mit Tabellen, etwa Verbstusstabellen im Bahnbereich. Die Integration solcher Techniken und ihre Transformation in formale Informatik-Methoden steckt noch in den Anfängen.

Metasprache für TTCN-3

TTCN-3 ist eine der ersten standardisierten Sprachen, die speziell für Testzwecke entworfen wurde. Sie ist allerdings sprachlich eher einer Programmiersprache als einer Modellierungssprache vergleichbar. Es werden daher mächtige Prinzipien und Werkzeuge gebraucht, um formale Spezifikations- und Modellierungssprachen in Sprachen wie TTCN-3 umzusetzen.

Formalisierungsmöglichkeit für Hardware/Software-Korrelationen

Im Bereich eingebetteter Steuerungen spielt nicht nur die Software eine Rolle, die Korrektheit hängt meist auch vom korrekten Zusammenspiel von Hardware- und Softwarekomponenten ab. Ob ein mobiler Roboter beispielsweise eine vorgegebene Bahn korrekt verfolgen kann, ist nicht nur eine Frage der Software, sondern auch der Toleranzen im Getriebe etc. Bislang existieren kaum Versuche, diese Korrelationen in formalen Spezifikationen mit zu berücksichtigen.

4.4 Mobile Systeme (UMTS und darüber hinaus)

Im Mobilfunk sind die technischen Grundlagen für die dritte Generation mobiler Geräte so gut wie fertig, in der Forschung denkt man über die vierte Generation nach. Demgegenüber sind die softwaretechnischen Auswirkungen und theoretischen Grundlagen des „mobilen Computing“ noch nicht genügend erforscht.

Kombination von Zeit und Raum

Modallogiken, die zeitliche und räumliche Dimensionen beinhalten, haben oft eine hohe Komplexität. Es ist nicht klar, was adäquate Formalisierungen relativistischer Raum/Zeit-Modelle sind und wie diese für die Spezifikation von Berechnungsmodellen genutzt werden können.

Description logics, hybrid logics

Ein aktuelles Gebiet der Logik ist die Erweiterung modaler oder temporaler Logiken um Namenskonzepte und andere erststufige Konstrukte. Interessant sind hier neben logischen Fragen wie Expressivität und Komplexität auch die Anwendungen zur Beschreibung zum Beispiel von mobilen Kommunikationsprotokollen.

Algorithmen

Für die genannten Logiken werden Algorithmen und Werkzeuge gebraucht, um sie in realen Anwendungen einsetzen zu können. Es ist nicht klar, inwieweit sich Partialordnungsverfahren und symbolische Modellprüfungsverfahren direkt übertragen lassen.

4.5 Tools

Wie schon mehrfach betont, hängt der Erfolg formaler Methoden ganz entscheidend von der Verfügbarkeit von zuverlässigen Werkzeugen ab, die auch dann von Entwicklern eingesetzt werden können, wenn sie keine Experten in der zugrunde liegenden Theorie sind. Dies kann zum Beispiel dadurch erreicht werden, dass die graphischen Anteile in den Benutzungsschnittstellen der Werkzeuge verstärkt werden.

Visuelle Spezifikationsverfahren

Ein relativ neuer Gedanke ist es, die einzugebende formale Spezifikation gänzlich mit Hilfe graphischer Eingabeverfahren zu definieren, zum Beispiel durch Interaktion mit einer entsprechenden 3D-Umgebung oder durch andere visuelle Verfahren. Für die von mir vorgestellten temporalen Logiken sind solche Überlegungen noch im Anfangsstadium.

Visualisierung von Test- und Wiederlegungssequenzen

Auf der Ausgabeseite sind visuelle Verfahren viel leichter zu konzipieren; hier ist eher die Frage nach generellen und effizienten Techniken für die Realisierung interessant. Mir schwebt hier mittelfristig eine standardisierte Anbindung von Validierungswerkzeugen an Simulationen in der Sprache „VRML“ (virtual reality modelling language) vor.

Verbindung von abstrakter Interpretation und Debugging

Hinsichtlich der Verbindung von formalen mit herkömmlichen Werkzeugen ist neben dem modellbasierten und codebasierten Testen vor allem das spezifikationsbasierte Debugging zu nennen. Es gibt bereits erste Ansätze zur Integration von temporallogischen Spezifikationen mit Debugging-Werkzeugen, für einen breiten Erfolg müssen die verwendeten abstrakten Interpretationen noch verbessert werden.

4.6 Security (Informationssicherheit)

Ein relativ junges Gebiet ist die Verwendung formaler Methoden für den Nachweis von Eigenschaften der Informationssicherheit. Auf Grund der momentan vorherrschenden Monopolisierung des Softwaremarktes und dem zunehmenden Einsatz elektronischer Systeme im Geschäftsverkehr wird die Bedeutung dieses Gebietes jedoch vermutlich noch sehr zunehmen.

Formalisierung von Unternehmenskommunikation

Nachdem besonders die Kommunikation zwischen Unternehmen immer mehr über Computernetze abgewickelt wird, ist die

Anwendung formaler Methoden zur Modellierung und Verifikation von Geschäftsprozessen ein spannendes neues Anwendungsfeld.

Modellprüfung für Protokolle und Verfahren

Für etliche in der Informationssicherheit relevante Eigenschaften bieten herkömmliche Spezifikationsformalismen überhaupt keine Ausdrucksmöglichkeit – zum Beispiel die Aussage „es ist praktisch unmöglich, das Passwort zu erraten“ lässt sich nur schwer formal hinschreiben, und noch schwerer automatisch beweisen. Die Hinzunahme quantitativer Aspekte könnte auch für Zuverlässigkeitsanalysen von Bedeutung werden.

„IT-Security-Check“

Durch die Vielzahl von Konfigurationsmöglichkeiten von IT-Systemen in Bezug auf die Informationssicherheit werden bereits heute wissensbasierte Verfahren dafür eingesetzt. Die Kombination der dabei verwendeten Deduktionsmethoden mit den modellbasierten Ansätzen könnte ein interessantes neues Gebiet werden. Erste Vorüberlegungen für gemeinsame Projekte mit Instituten und Firmen haben bereits begonnen.

5 Forschung, Lehre und Transfer

Lassen Sie mich zum Schluss noch ein paar Worte über das Verhältnis von Forschung und Lehre zu industriellen Anwendungen sagen. Als Assistent an der TU München habe ich mich im „klassischen Stil“ der Lehre und Grundlagenforschung gewidmet, während ich als Geschäftsführer des Bremer Instituts für Sichere Systeme überwiegend mit Entwicklungen und anwendungsorientierter Forschung befasst war. Mein Credo aus diesen Erfahrungen ist es, dass heutzutage das Humboldtsche Ideal der Einheit von Forschung und Lehre um eine dritte Komponente erweitert werden muss, nämlich den Transfer in Anwendungen. Dies gilt – unabhängig von der beständigen Forderung der Sparpolitiker nach höheren Drittmittelzahlen – besonders für neue Fächer wie die Informatik, die sehr stark durch aktuelle gesell-

schaftliche und industrielle Anforderungen geprägt werden. Zwischen den drei Bestandteilen herrscht eine starke Interdependenz (Abb. 20):

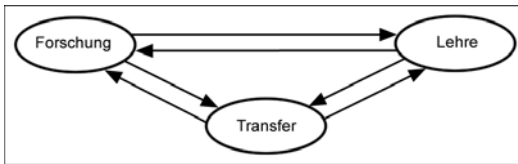


Abb. 20:
Interdependenz zwischen Forschung, Lehre und Transfer

- Dass sich Forschung und Lehre gegenseitig beflügeln, ist längst eine allgemein akzeptierte Tatsache. Einerseits ist es für die universitäre Lehre geradezu eine Notwendigkeit, dass sie ständig durch neue Forschungsergebnisse aktualisiert wird. Andererseits wird man als Forscher durch die Präsentation in Lehrveranstaltungen und die kritischen Fragen der Studierenden gezwungen, seine Gedanken klar zu ordnen und Sachverhalte verständlich darzustellen.
- Der Anstoß für neue industrielle Entwicklungen sind sehr häufig Forschungsergebnisse aus den Universitäten und Forschungsinstituten.
- Besonders in der Informatik ergeben sich interessante Forschungsthemen erst aus den Fragestellungen der Anwender.
- Die Ausbildung von Studierenden wird deutlich verbessert, wenn sie von Beispielen aus der Praxis begleitet wird.
- Oft sind studentische Ideen nicht von veralteten technologischen Vorurteilen beeinflusst und lassen sich vorteilhaft in Anwendungsprojekte integrieren.

Aus diesen Gründen ist die Position, die ich jetzt als Stiftungsprofessor an der Humboldt-Universität zu Berlin und Bereichsleiter am Fraunhofer-Institut bekleide, für mich besonders reizvoll. Mein Ziel ist der Aufbau eines Kompetenzzentrums für Softwarequalität, wobei ich die Grundlagenforschung innerhalb des Instituts für Informatik und die industrienahen Projekte im Fraunhofer-Verbund durchführen möchte. In diesem Sinne bedanke ich mich für Ihre Aufmerksamkeit und wünsche mir und Ihnen für die Zukunft eine gute Zusammenarbeit!

Anmerkungen

- [1] Computers in Spaceflight: The NASA Experience. <http://www.hq.nasa.gov/office/pao/History/computers/Ch2-6.html>; siehe auch <http://www.mech.uq.edu.au/courses/mech2700/mohler.html>; im Druck: Apollo Expeditions to the Moon, edited by Edgar M. Cortright, NASA SP; 350, Washington, DC, 1975.
- [2] Ariane 5 Flight 501 Failure: Report by the Inquiry Board, J.L. Lions, chairman; http://www.esa.int/export/esaCP/Pr_33_1996_p_EN.html.
- [3] Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions. JPL Special Review Board, Jet Propulsion Laboratory, California Institute of Technology, JPL D-18709, 22 March 2000; <http://mars.jpl.nasa.gov/msp98/>.
- [4] NPD 8730.4: NASA Policy directive 8730.4, Software Independent Verification and Validation (IV&V) Policy, 01 August 2001; <http://www.ivv.nasa.gov>.
- [5] Formal Methods Home Page: <http://www.afm.sbu.ac.uk/>.
- [6] DMS-R: ESA's Data Management System for the Russian Segment of the ISS http://www.esa.int/export/esaHS/ESA0XX0VMOC_iss_0.html.
- [7] Physikalisch-Technische Bundesanstalt (PTB): Wissenswertes zur Zeit http://www.ptb.de/de/blickpunkt/_infoszurzeit.html.
- [8] *Berthomieu, B. and Diaz, M.*: Modeling and Verification of Time Dependent Systems Using Time Petri Nets. IEEE Transactions on Software Engineering, Vol. 17, No. 3, pp 259–273. March 1991.
- [9] *Davies, J. W., Jackson, D. M., Reed, G. M., Reed, J. N., Roscoe, A. W. and Schneider, S. A.*: Timed CSP: Theory and Applications. Springer LNCS 600, pp 640–675, 1992; Proceedings of the REX Workshop, Nijmegen, 1991.
- [10] *Hoare, C. A. R.*: Communicating Sequential Processes. Prentice Hall International Series in Computer Science, 1985.
- [11] Formal Systems Ltd.: The FDR2 Model Checker. <http://www.fsel.com/>.
- [12] *Clarke, E. M., Grumberg, O. and Peled, D.*: Model Checking. MIT Press 1999.
- [13] RT-TOOLS: Workshop on Real-Time Tools; <http://user.it.uu.se/~paupet/pc/rttools-2002/>
- [14] *Valmari, A.*: A stubborn attack on state explosion. In Proceedings of CAV'90, pp 25–42. ACM, DIMACS volume 3, 1990.
- [15] *Bengtsson, J., Jonsson, B., Lilius, J. and Yi, W.*: Partial Order Reduc-

- tions for Timed Systems In: CONCUR '98, International Conference on Concurrency Theory, LNCS 1466, 1998.
- [16] *Dams, D., Gerth, R., Knaack, B. and Kuiper, R.*: Partial-order Reduction Techniques for Real-time Model Checking. *Formal Aspects of Computing* 10(5–6): 469–482, 1998.
 - [17] *Bieber, B. und Fleischhack, H.*: Model Checking of Time Petri Nets Based on Partial Order Semantics. In J.C.M. Baeten and S. Mauw, editors, CONCUR '99, International Conference on Concurrency Theory, LNCS 1664, pp 210–225, 1999.
 - [18] *Minea, M.*: Partial Order Reduction for Verification of Timed Systems. Ph.D. Thesis, Carnegie Mellon University, Report CMU-CS-00-102, 124 pp, Dec. 1999.
 - [19] UPPAAL: The Uppsala and Aalborg Universities model checker; <http://www.uppaal.com/>.
 - [20] *Dill, D.*: Trace Theory for Automatic Hierarchical Verification of Speed Independent Circuits; Diss. CMU SCS, MIT Press 1988.
 - [21] Verified Systems International GmbH, RT-Tester. <http://www.verified.de/rt-tester.html>.
 - [22] 3rd Generation Partnership Project: UMTS Specification. <http://www.3gpp.org/>.

Bernd-Holger Schlingloff

- 1959 geboren in Berlin.
- 1965–1968 Grundschule in Berlin und Göttingen.
- 1969–1977 Gymnasium in Kiel und München.
- 1977 Abitur am Rupprecht-Gymnasium, München.
- 1977–1978 Studium der Philosophie und Logik an der Ludwig-Maximilians-Universität München.
- 1978–1984 Studium der Informatik an der Technischen Universität München.
- 1984 Diplom Informatik, Thema der Diplomarbeit „Beweistheoretische Untersuchungen zur temporalen Logik“, Prof. F. Kröger.
- 1984–1990 wissenschaftlicher Angestellter am Institut für Informatik der Technischen Universität München.
- 1990 Promotion in Informatik (Dr. rer. nat.) mit einer Arbeit zur temporalen Logik von Bäumen.
- 1991 Gastwissenschaftler an der School of Computer Science der Carnegie-Mellon Universität, Pittsburgh, Prof. E.M. Clarke und Prof. D. Scott.
- 1992–1996 wissenschaftlicher Assistent am Institut für Informatik der Technischen Universität München, Prof. M. Paul.
- 1996–2001 Geschäftsführer des Bremer Instituts für Sichere Systeme (BISS) am Technologie-Zentrum Informatik (TZI) der Universität Bremen.
- 2001 Habilitation für Informatik (Dr. rer. nat. habil.), Thema der Habilitationsschrift „Partial State Space Analysis of Safety-Critical Systems“.
- 2001–2002 Oberassistent am Fachbereich 3 (Mathematik/Informatik) der Universität Bremen, Prof. B. Krieg-Brückner.
- Seit 2002 Professor für Informatik an der Humboldt-Universität zu Berlin sowie Bereichsleiter am Fraunhofer-Institut für Rechnerarchitektur und Softwaretechnik FIRST Berlin.

Ausgewählte Veröffentlichungen

- *M. Frey, H. Schlingloff*: Conformance of Distributed Systems; In „TestCom 2003“. Kluwer Academic Publishers (May 2003).
- *J. Brederke, H. Schlingloff*: An Automated, Flexible Testing Environment for UMTS; In „Testing Internet Technologies and Services“. Kluwer Academic Publishers (2002).
- *E.M. Clarke, H. Schlingloff*: Model Checking; Chapter 21 in Alan Robinson and Andrei Voronkov (eds.), Handbook of Automated Reasoning; Elsevier Science Publishers B.V., pp. 1367–1522 (2000).
- *H. Schlingloff, O. Meyer, Th. Hülsing*: Correctness Analysis of an Embedded Controller; Proc. Int. Conf. on Data Systems in Aerospace (DASIA 99), Lissabon (May 1999).
- *I. Honma, H. Schlingloff, T. Yoneda*: Verification of bounded delay asynchronous circuits with timed traces; Proc. 7th Int. Conf. on Algebraic Methodology and Software Technology (AMAST'98), Amazonia, Brazil, Springer LNCS (Jan. 1999).
- *L. Twele, H. Schlingloff, H. Szczerbicka*: Performability Analysis of an Avionics-Interface; Proc. IEEE Conf. on Systems, Man and Cybernetics; San Diego, N.J., pp. 499–504, (Oct. 1998).
- *T. Yoneda, H. Schlingloff*: Efficient Verification of Parallel Real-Time Systems; Journal of Formal Methods in System Design 11-2, pp. 187–215, (1997).
- *H. Schlingloff, W. Heinle*: Relation Algebra and Modal Logics; Relational Methods in Computer Science, C. Brink, W. Kahl and G. Schmidt (eds), pp. 70–89, Springer Advances in Computing (1997).
- *H. Schlingloff*: On the Expressive Power of Modal Logic on Trees; Proc. Int. Conf. on Logical Foundations of Computer Science (LFCS '92), Logic at Tver, Springer LNCS 620, pp. 441–451 (1992).
- *H. Schlingloff*: Expressive Completeness of Temporal Logic of Trees; Journal of Applied Non-Classical Logics, Vol.2-2, pp. 157–180 (1992).

In der Reihe **Öffentliche Vorlesungen** sind erschienen:

- | | | | | | |
|----|--|----|--|----|---|
| 1 | <i>Volker Gerhardt</i>
Zur philosophischen Tradition der Humboldt-Universität | 14 | <i>Ludolf Herbst</i>
Der Marshallplan als Herrschaftsinstrument?
Überlegungen zur Struktur amerikanischer Nachkriegspolitik | 26 | <i>Ludmila Thomas</i>
Rußland im Jahre 1900
Die Gesellschaft vor der Revolution |
| 2 | <i>Hasso Hofmann</i>
Die versprochene Menschenwürde | 15 | <i>Gert-Joachim Glaeßner</i>
Demokratie nach dem Ende des Kommunismus | 27 | <i>Wolfgang Reisig</i>
Verteiltes Rechnen: Im wesentlichen das Herkömmliche oder etwas grundlegend Neues? |
| 3 | <i>Heinrich August Winkler</i>
Von Weimar zu Hitler
Die Arbeiterbewegung und das Scheitern der ersten deutschen Demokratie | 16 | <i>Arndt Sorge</i>
Arbeit, Organisation und Arbeitsbeziehungen in Ostdeutschland | 28 | <i>Ernst Osterkamp</i>
Die Seele des historischen Subjekts
Historische Portraituren in Friedrich Schillers „Geschichte des Abfalls der vereinigten Niederlande von der Spanischen Regierung“ |
| 4 | <i>Michael Borgolte</i>
„Totale Geschichte“ des Mittelalters?
Das Beispiel der Stiftungen | 17 | <i>Achim Leube</i>
Semnonen, Burgunden, Alamannen
Archäologische Beiträge zur germanischen Frühgeschichte des 1. bis 5. Jahrhunderts | 29 | <i>Rüdiger Steinlein</i>
Märchen als poetische Erziehungsform
Zum kinderlandliterarischen Status der Grimmschen „Kinder- und Hausmärchen“ |
| 5 | <i>Wilfried Nippel</i>
Max Weber und die Althistorie seiner Zeit | 18 | <i>Klaus-Peter Johné</i>
Von der Kolonienwirtschaft zum Kolonat
Ein römisches Abhängigkeitsverhältnis im Spiegel der Forschung | 30 | <i>Hartmut Boockmann</i>
Bürgerkirchen im späteren Mittelalter |
| 6 | <i>Heinz Schilling</i>
Am Anfang waren Luther, Loyola und Calvin – ein religionssoziologisch-entwicklungsgeschichtlicher Vergleich | 19 | <i>Volker Gerhardt</i>
Die Politik und das Leben | 31 | <i>Michael Kloepfer</i>
Verfassungsgebung als Zukunftsbewältigung aus Vergangenheitserfahrung
Zur Verfassungsgebung im vereinten Deutschland |
| 7 | <i>Hartmut Harnisch</i>
Adel und Großgrundbesitz im ostelbischen Preußen 1800–1914 | 20 | <i>Clemens Wurm</i>
Großbritannien, Frankreich und die westeuropäische Integration | 32 | <i>Dietrich Benner</i>
Über die Aufgaben der Pädagogik nach dem Ende der DDR |
| 8 | <i>Fritz Jost</i>
Selbststeuerung des Justizsystems durch richterliche Ordnungen | 21 | <i>Jürgen Kunze</i>
Verbiefeldstrukturen | 33 | <i>Heinz-Elmar Tenorth</i>
„Reformpädagogik“
Erneuter Versuch, ein erstaunliches Phänomen zu verstehen |
| 9 | <i>Erwin J. Haeberle</i>
Berlin und die internationale Sexualwissenschaft
Magnus Hirschfeld-Kolloquium, Einführungsvortrag | 22 | <i>Winfried Schich</i>
Die Havel als Wasserstraße im Mittelalter: Brücken, Dämme, Mühlen, Flutrinnen | 34 | <i>Jürgen K. Schriewer</i>
Welt-System und Interrelations-Gefüge
Die Internationalisierung der Pädagogik als Problem Vergleichender Erziehungswissenschaft |
| 10 | <i>Herbert Schnädelbach</i>
Hegels Lehre von der Wahrheit | 23 | <i>Herfried Münkler</i>
Zivilgesellschaft und Bürgertugend
Bedürfen demokratisch verfaßte Gemeinwesen einer sozio-moralischen Fundierung? | 35 | <i>Friedrich Maier</i>
„Das Staatsschiff“ auf der Fahrt von Griechenland über Rom nach Europa
Zu einer Metapher als Bildungsgegenstand in Text und Bild |
| 11 | <i>Felix Herzog</i>
Über die Grenzen der Wirksamkeit des Strafrechts
Eine Hommage an Wilhelm von Humboldt | 24 | <i>Hildegard Maria Nickel</i>
Geschlechterverhältnis in der Wende
Individualisierung versus Solidarisierung? | 36 | <i>Michael Daxner</i>
Alma Mater Restituta oder Eine Universität für die Hauptstadt |
| 12 | <i>Hans-Peter Müller</i>
Soziale Differenzierung und Individualität
Georg Simmels Gesellschafts- und Zeitdiagnose | 25 | <i>Christine Windbichler</i>
Arbeitsrechtler und andere Laien in der Baugrube des Gesellschaftsrechts
Rechtsanwendung und Rechtsfortbildung | | |
| 13 | <i>Thomas Raiser</i>
Aufgaben der Rechtssoziologie als Zweig der Rechtswissenschaft | | | | |

- 37 *Konrad H. Jarausch*
Die Vertreibung der jüdischen Studenten und Professoren von der Berliner Universität unter dem NS-Regime
- 38 *Detlef Krauß*
Schuld im Strafrecht
Zurechnung der Tat oder Abrechnung mit dem Täter?
- 39 *Herbert Kitschelt*
Rationale Verfassungswahl?
Zum Design von Regierungssystemen in neuen Konkurrenzdemokratien
- 40 *Werner Röcke*
Liebe und Melancholie
Formen sozialer Kommunikation in der ‚Historie von Florio und Blanschefleur‘
- 41 *Hubert Markl*
Wohin geht die Biologie?
- 42 *Hans Bertram*
Die Stadt, das Individuum und das Verschwinden der Familie
- 43 *Dieter Segert*
Diktatur und Demokratie in Osteuropa im 20. Jahrhundert
- 44 *Klaus R. Scherpe*
Beschreiben, nicht Erzählen!
Beispiele zu einer ästhetischen Opposition: Von Döblin und Musil bis zu Darstellungen des Holocaust
- 45 *Bernd Wegener*
Soziale Gerechtigkeitsforschung: Normativ oder deskriptiv?
- 46 *Horst Wenzel*
Hören und Sehen – Schrift und Bild
Zur mittelalterlichen Vorgeschiede audiovisueller Medien
- 47 *Hans-Peter Schwintowski*
Verteilungsdefizite durch Recht auf globalisierten Märkten
Grundstrukturen einer Nutzentheorie des Rechts
- 48 *Helmut Wiesenthal*
Die Krise holistischer Politikansätze und das Projekt der gesteuerten Systemtransformation
- 49 *Rainer Dietrich*
Wahrscheinlich regelhaft. Gedanken zur Natur der inneren Sprachverarbeitung
- 50 *Bernd Henningsen*
Der Norden: Eine Erfindung
Das europäische Projekt einer regionalen Identität
- 51 *Michael C. Burda*
Ist das Maß halb leer, halb voll oder einfach voll?
Die volkswirtschaftlichen Perspektiven der neuen Bundesländer
- 52 *Volker Neumann*
Menschenwürde und Existenzminimum
- 53 *Wolfgang Iser*
Das Großbritannien-Zentrum in kulturwissenschaftlicher Sicht
Vortrag anlässlich der Eröffnung des Großbritannien-Zentrums an der Humboldt-Universität zu Berlin
- 54 *Ulrich Battis*
Demokratie als Bauherrin
- 55 *Johannes Hager*
Grundrechte im Privatrecht
- 56 *Johannes Christes*
Cicero und der römische Humanismus
- 57 *Wolfgang Hardtwig*
Vom Elitebewußtsein zur Massenbewegung – Frühformen des Nationalismus in Deutschland 1500–1840
- 58 *Elard Klewitz*
Sachunterricht zwischen Wissenschaftsorientierung und Kindbezug
- 59 *Renate Valtin*
Die Welt mit den Augen der Kinder betrachten
Der Beitrag der Entwicklungstheorie Piagets zur Grundschulpädagogik
- 60 *Gerhard Werle*
Ohne Wahrheit keine Versöhnung!
Der südafrikanische Rechtsstaat und die Apartheid-Vergangenheit
- 61 *Bernhard Schlink*
Rechtsstaat und revolutionäre Gerechtigkeit. Vergangenheit als Zumutung?
(Zwei Vorlesungen)
- 62 *Wiltrud Gieseke*
Erfahrungen als behindernde und fördernde Momente im Lernprozeß Erwachsener
- 63 *Alexander Demandt*
Ranke unter den Weltweisen
Wolfgang Hardtwig
Die Geschichtserfahrung der Moderne und die Ästhetisierung der Geschichtsschreibung: Leopold von Ranke
(Zwei Vorträge anlässlich der 200. Wiederkehr des Geburtstages Leopold von Rankes)
- 64 *Axel Flessner*
Deutsche Juristenausbildung
Die kleine Reform und die europäische Perspektive
- 65 *Peter Brockmeier*
Seul dans mon lit glacé – Samuel Becketts Erzählungen vom Unbehagen in der Kultur
- 66 *Hartmut Böhme*
Das Licht als Medium der Kunst
Über Erfahrungsarmut und ästhetisches Gegenlicht in der technischen Zivilisation
- 67 *Siegling Ellger-Rüttgardt*
Berliner Rehabilitationspädagogik: Eine pädagogische Disziplin auf der Suche nach neuer Identität
- 68 *Christoph G. Paulus*
Rechtsgeschichtliche und rechtsvergleichende Betrachtungen im Zusammenhang mit der Beweisvereitelung
- 69 *Eberhard Schwark*
Wirtschaftsordnung und Sozialstaatsprinzip
- 70 *Rosemarie Will*
Eigentumstransformation unter dem Grundgesetz
- 71 *Achim Leschinsky*
Freie Schulwahl und staatliche Steuerung
Neue Regelungen des Übergangs an weiterführende Schulen
- 72 *Harry Dettenborn*
Hang und Zwang zur sozial-kognitiven Komplexitätsreduzierung: Ein Aspekt moralischer Urteilsprozesse bei Kindern und Jugendlichen
- 73 *Inge Frohburg*
Blickrichtung Psychotherapie: Potenzen – Realitäten – Folgerungen
- 74 *Johann Adrian*
Patentrecht im Spannungsfeld von Innovationsschutz und Allgemeininteresse

- 75 *Monika Doherty*
Verständigung trotz allem. Probleme aus und mit der Wissenschaft vom Übersetzen
- 76 *Jürgen van Buer*
Pädagogische Freiheit, pädagogische Freiräume und berufliche Situation von Lehrern an Wirtschaftsschulen in den neuen Bundesländern
- 77 *Flora Veit-Wild*
Karneval und Kakerlaken
Postkolonialismus in der afrikanischen Literatur
- 78 *Jürgen Diederich*
Was lernt man, wenn man nicht lernt? Etwas Didaktik „jenseits von Gut und Böse“ (Nietzsche)
- 79 *Wolf Krötke*
Was ist „wirklich“?
Der notwendige Beitrag der Theologie zum Wirklichkeitsverständnis unserer Zeit
- 80 *Matthias Jerusalem*
Die Entwicklung von Selbstkonzepten und ihre Bedeutung für Motivationsprozesse im Lern- und Leistungsbereich
- 81 *Dieter Klein*
Globalisierung und Fragen an die Sozialwissenschaften: Richtungsbestimmter Handlungszwang oder Anstoß zu einschneidendem Wandel?
- 82 *Barbara Kunzmann-Müller*
Typologisch relevante Variation in der Slavia
- 83 *Michael Parmentier*
Sehen Sehen
Ein bildungstheoretischer Versuch über Chardins ‚L'enfant au toton‘
- 84 *Engelbert Plassmann*
Bibliotheksgeschichte und Verfassungsgeschichte
- 85 *Ruth Tesmar*
Das dritte Auge
Imagination und Einsicht
- 86 *Orfried Schöffler*
Perspektiven erwachsenenpädagogischer Organisationsforschung
- 87 *Kurt-Victor Selge, Reimer Hansen, Christof Gestrich*
Philipp Melanchthon 1497–1997
- 88 *Karla Horstmann-Hegel*
Integrativer Sachunterricht – Möglichkeiten und Grenzen
- 89 *Karin Hirdina*
Belichten. Beleuchten. Erhellen
Licht in den zwanziger Jahren
- 90 *Marion Bergk*
Schreibinteraktionen: Verändertes Sprachlernen in der Grundschule
- 91 *Christina von Braun*
Architektur der Denkräume
James E. Young
Daniel Libeskind's Jewish Museum in Berlin: The Uncanny Art of Memorial Architecture
Daniel Libeskind
Beyond the Wall
Vorträge anlässlich der Verleihung der Ehrendoktorwürde an Daniel Libeskind
- 92 *Christina von Braun*
Warum Gender-Studies?
- 93 *Ernst Vogt, Axel Horstmann*
August Boeckh (1785–1867). Leben und Werk
Zwei Vorträge
- 94 *Engelbert Plassmann*
Eine „Reichsbibliothek“?
- 95 *Renate Reschke*
Die Asymmetrie des Ästhetischen
Asymmetrie als Denkfigur historisch-ästhetischer Dimension
- 96 *Günter de Bruyn*
Altersbetrachtungen über den alten Fontane
Festvortrag anlässlich der Verleihung der Ehrendoktorwürde
- 97 *Detlef Krauß*
Gift im Strafrecht
- 98 *Wolfgang Thierse, Renate Reschke, Achim Trebeß, Claudia Salchow*
Das Wolfgang-Heise-Archiv. Plädoyers für seine Zukunft
Vorträge
- 99 *Elke Lehnert, Annette Vogt, Ulla Ruschhaupt, Marianne Kriszto*
Frauen an der Humboldt-Universität 1908–1998
Vier Vorträge
- 100 *Bernhard Schlink*
Evaluierte Freiheit?
Zu den Bemühungen um eine Verbesserung der wissenschaftlichen Lehre
- 101 *Heinz Ohme*
Das Kosovo und die Serbische Orthodoxe Kirche
- 102 *Gerhard A. Ritter*
Der Berliner Reichstag in der politischen Kultur der Kaiserzeit
Festvortrag anlässlich der Verleihung der Ehrendoktorwürde mit einer Laudatio von Wolfgang Hardtwig
- 103 *Cornelius Frömmel*
Das Flair der unendlichen Vielfalt
- 104 *Verena Olejniczak Lobsien*
„Is this the promised end?“ Die Apokalypse des King Lear, oder: Fängt Literatur mit dem Ende an?
- 105 *Ingolf Pernice*
Kompetenzabgrenzung im Europäischen Verfassungsverbund
- 106 *Gerd Irrlitz*
Das Bild des Weges in der Philosophie
- 107 *Helmut Schmidt*
Die Selbstbehauptung Europas im neuen Jahrhundert. Mit einer Replik von Horst Teltshik
- 108 *Peter Diepold*
Internet und Pädagogik
Rückblick und Ausblick
- 109 *Artur-Axel Wandtke*
Copyright und virtueller Markt oder Das Verschwinden des Urhebers im Nebel der Postmoderne?
- 110 *Jürgen Mittelstraß*
Konstruktion und Deutung
Über Wissenschaft in einer Leonardo- und Leibniz-Welt
- 111 *Göran Persson*
European Challenges. A Swedish Perspective. Mit einer Replik von Janusz Reiter
- 112 *Hasso Hofmann*
Vom Wesen der Verfassung
- 113 *Stefanie von Schurbein*
Kampf um Subjektivität
Nation, Religion und Geschlecht in zwei dänischen Romanen um 1850

- 114 *Ferenc Mád*
Europäischer Integrationsprozess. Ungarische Erwartungen. Mit einer Replik von Dietrich von Kyaw
- 115 *Ernst Maug*
Konzerne im Kontext der Kapitalmärkte
- 116 *Herbert Schnädelbach*
Das Gespräch der Philosophie
- 117 *Axel Flessner*
Juristische Methode und europäisches Privatrecht
- 118 *Sigrid Jacobeit*
KZ-Gedenkstätten als nationale Erinnerungsorte
Zwischen Ritualisierung und Musealisierung
- 119 *Vincent J. H. Houben*
Südostasien. Eine andere Geschichte
- 120 *Étienne Balibar, Friedrich A. Kittler, Martin van Creveld*
Vom Krieg zum Terrorismus?
Mosse-Lectures 2002/2003
- 121 *Hans Meyer*
Versuch über die Demokratie in Deutschland
- 122 *Joachim Kallinich*
Keine Atempause – Geschichte wird gemacht
Museen in der Erlebnis- und Mediengesellschaft
- 123 *Anusch Taraz*
Zufällige Beweise
- 124 *Carlo Azeglio Ciampi*
L'amicizia italo-tedesca al servizio dell'integrazione europea. Die italienisch-deutsche Freundschaft im Dienste der europäischen Integration
Johannes Rau
Deutschland, Italien und die europäische Integration
- 125 *Theodor Schilling*
Der Schutz der Menschenrechte gegen den Sicherheitsrat und seine Mitglieder
Möglichkeiten und Grenzen
- 126 *Wolfgang Ernst*
Medienwissen(schaft) zeitkritisch
Ein Programm aus der Sophienstraße
- 127 *Hilmar Schröder*
Klimawärmung und Naturkatastrophen im Hochgebirge
Desaster oder Stabilität im 21. Jahrhundert?
- 128 *Kiran Klaus Patel*
Nach der Nationalfixiertheit
Perspektiven einer transnationalen Geschichte
- 129 *Susanne Frank*
Stadtplanung im Geschlechterkampf
Ebenezer Howard und Le Corbusier
- 130 *Matthias Langensiepen*
Modellierung pflanzlicher Systeme
Perspektiven eines neuen Forschungs- und Lehrgebietes
- 131 *Michael Borgolte*
Königsberg – Deutschland – Europa
Heinrich August Winkler und die Einheit der Geschichte. Festvortrag anlässlich des 65. Geburtstages
- 132 *Guy Verhofstadt*
The new European Constitution – from Laeken to Rome
- 133 *Elke Hartmann*
Zur Geschichte der Matriarchatsidee
- 134 *Felix Naumann*
Informationsintegration
- 135 *Gerhard Dannemann*
Rechtsvergleichung im Exil
Martin Wolff und das englische Recht
- 136 *Jörg Baberowski*
Zivilisation der Gewalt
Die kulturellen Ursprünge des Stalinismus
- 137 *Friedhelm Neidhardt*
Logik – Soziologik
Kolloquium anlässlich der Verleihung der Ehrendoktorwürde
- 138 *Bernd-Holger Schlingloff*
Formale Methoden in der Praxis
Softwaredesign für Luft- und Raumfahrt
- 139 *Sigrid Blömeke*
Lehrerausbildung – Lehrerhandeln – Schülerleistungen
Perspektiven nationaler und internationaler empirischer Bildungsforschung
- 140 *Katharina Bracht*
Securitas libertatis
Augustins Entdeckung der radikalen Entscheidungsfreiheit als Ursprung des Bösen
- 141 *Friedrich Dieckmann*
Berlin als Werkraum
Stadtuldigung mit Seitenblicken
Festvortrag anlässlich der Verleihung der Ehrendoktorwürde